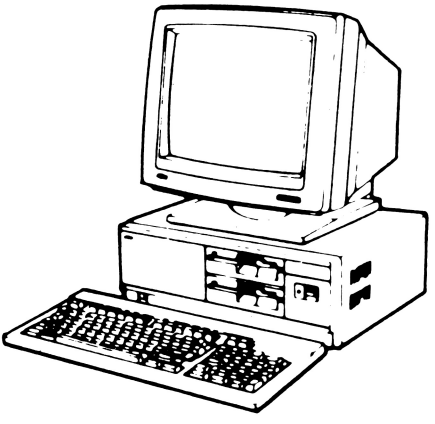
APC III



MS™-DOS
Macro Assembler Manual
Volume 2

NEC
NEC Information Systems, Inc.
819-150016-000 Rev. 00
5-84

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the Microsoft Macro Assembler Manual on magnetic tape, disk, or any other medium for any purpose other than the purchaser's personal use.

(C) Microsoft Corporation 1981, 1983

Comments about this documentation may be sent to:

Microsoft Corporation
Microsoft Building
10700 Northup Way
Bellevue, WA 98004

Microsoft is a registered trademark of Microsoft Corporation.

MS is a trademark of Microsoft Corporation.

Intel is a trademark of Intel Corporation.

Document Number: 8407-200-02
Part Number: 016-014-005

APC III

NEC INFORMATION SYSTEMS PERSONAL COMPUTER
LIMITED WARRANTY
SOFTWARE REGISTRATION CARD

PLEASE READ THE FOLLOWING TEXT CAREFULLY. IT CONSTITUTES A CONTINUATION OF THE PROGRAM LICENSE AGREEMENT FOR THE SOFTWARE APPLICATION PROGRAM CONTAINED IN THIS PACKAGE.

If you agree to all the terms and conditions contained in both parts of the Program License Agreement, please fill out the detachable postcard and return it to:

NEC Information Systems, Inc.
1414 Massachusetts Ave.
Boxborough, MA 01719

Attention: Customer Technical Services

LIABILITY

In no event shall the copyright holder, the original licensor nor any intermediate sublicensors of this software be responsible for any indirect or consequential damages or lost profits arising from the use of this software.

COPYRIGHT

The name of the copyright holder of this software must be recorded exactly as it appears on the label of the original diskette as supplied by NECIS on a label attached to each additional copy you make.

You must maintain a record of the number and location of each copy of this program.

All NECIS software programs and copies remain the property of the copyright holder, though the physical medium on which they exist is the property of the licensee.

MERGING, ALTERATION

Should this program be merged with or incorporated into another program, or altered in any way by the licensee, the terms of the Warranty contained herein are voided and neither NECIS nor the copyright holder nor any intermediate sublicensors will assure the conformity of this software to its specification nor refund the license fee for such nonconformity.

Upon termination of this license for any reason, any such merged or incorporated programs must be separated from the programs with which they have been merged or incorporated and any altered programs must be destroyed.

Please fill in this form completely and carefully. Failure to do so will terminate your license agreement for this software.

When you have completed this form, detach it from the rest of the Program License Agreement, and drop in any mailbox. The remaining portion of the license agreement should be retained for your records.

819-000100-6D01 Rev. 01

"The undersigned Enduser of this NECIS software hereby acknowledges that he/she has read and fully understands the terms and conditions of this license agreement, hereby incorporated into this card and acknowledged by this reference.

The undersigned hereby acknowledges that by signing this document, he/she becomes a party to said license with NECIS, and agrees to be bound by all terms, conditions, and obligations contained therein."

PROGRAM NAME (as it appears on diskette label):

_____

SERIAL NUMBER: _____ DATE PURCHASED: _____

DEALER NAME: _____ CITY: _____ STATE: _____

YOUR NAME: _____

YOUR COMPANY (only if company purchase): _____

YOUR ADDRESS: _____

YOUR CITY, STATE, ZIP: _____

YOUR TELEPHONE: _____

"Warranty Requires Return of This Card" 819-000100-6D01 Rev. 01

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 105 BOXBOROUGH, MA

POSTAGE WILL BE PAID BY ADDRESSEE

NEC Information Systems, Inc.
1414 Massachusetts Ave.
Boxborough, MA 01719

Attention: Customer Technical Services

Contents

2 disks, with the following files:

MASM.EXE
LINK.EXE
CREF.EXE
DEBUG.EXE

LIB.EXE

1 binder (titled Microsoft Macro Assembler Manual) with 5 manuals:

Microsoft Macro Assembler Utility Manual
Microsoft LINK Linker Utility Manual
Microsoft LIB Library Manager Manual
Microsoft CREF Cross-Reference Utility Manual
Microsoft DEBUG Utility Manual

System Requirements

Each utility requires different amounts of memory.

Macro Assembler - 96K bytes of memory minimum:
64K bytes for code and static data
32K bytes for run space

Microsoft LINK - 50K bytes of memory minimum:
40K bytes for code
10K bytes for run space

Microsoft LIB - 38K bytes of memory minimum:
28K bytes for code
10K bytes for run space

Microsoft CREF - 24K bytes of memory minimum:
14K bytes for code
10K bytes for run space

Microsoft DEBUG - Memory minimum program-dependent
13K bytes for code
Run space program-dependent

Disk drive(s)
One disk drive if and only if output is sent to the same physical disk from which the input was taken. None of the utility programs allows time to swap disks during operation on a one-drive configuration. Therefore, two disk drives is a more practical configuration.

Microsoft

Welcome to the Microsoft(R) family of products.

Microsoft Corporation continues to supply consistently high-quality software for all types of users.

In addition to the Macro Assembler and Microsoft BASIC interpreter, Microsoft sells other full-feature language compilers, language subsets, and operating system products. Microsoft offers a "family" of software products that both look alike from one product to the next, and can be used together for effective program development.

For more information about other Microsoft products, contact:

Microsoft Corporation
10700 Northup Way
Bellevue, WA 98004
(206) 828-8080

Contents

General Introduction

Major Features
Using These Manuals
Syntax Notation
Learning More about Assembly Language Programming
Overview of Program Development

Microsoft Macro Assembler Utility

Introduction

Chapter 1 Creating a Macro Assembler Source File

Chapter 2 Names: Labels, Variables, and Symbols

Chapter 3 Expressions: Operands and Operators

Chapter 4 Action: Instructions and Directives

Chapter 5 Assembling a Macro Assembler Source File

Chapter 6 8087 Support

Chapter 7 Macro Assembler Messages

Appendices

Index for Macro Assembler

Microsoft LINK Linker Utility

Chapter 1 Introduction

Chapter 2 MS-LINK Technical Information

Index for MS-LINK

Microsoft LIB Library Manager

Chapter 1 Introduction

Chapter 2 Running MS-LIB

Chapter 3 Error Messages

Index for MS-LIB

Microsoft CREF Cross Reference Utility

Chapter 1 Introduction

Chapter 2 Running MS-CREF

Chapter 3 Error Messages

Chapter 4 Format of MS-CREF Compatible Files

Index for MS-CREF

Microsoft DEBUG Utility

Chapter 1 Introduction

Chapter 2 Commands

Index for DEBUG

The Microsoft Macro Assembler Manual includes utility programs used for developing assembly language programs. In addition, the Microsoft LINK Linker Utility and DEBUG are used with of Microsoft's 16-bit language compilers.

Major Features

Macro Assembler Utility

Microsoft's Macro Assembler is a powerful assembler for 8086 based computers.

Macro Assembler supports most of the directives found in Microsoft's Macro Assembler for the 8080. Macros and conditionals are Intel 8080 standard.

Macro Assembler is upward compatible with Intel's ASM-86, except Intel codemacros, macros, and a few $ directives.

Macro Assembler offers relaxed typing so that if you enter a typeless operand for an instruction that accepts only one type of operand, Macro Assembler assembles the statement correctly instead of returning an error message.

Microsoft LINK Linker Utility (Technical Information Only)

MS-LINK is a virtual linker, which can link programs that are larger than available memory.

MS-LINK produces relocatable executable object code.

MS-LINK processes overlays that you define.

MS-LINK can perform multiple library searches, using a dictionary library search method.

MS-LINK prompts you for input and output modules and other link session parameters.

MS-LINK can be run with an automatic response file to answer the Linker prompts.

Microsoft LIB Library Manager

MS-LIB can add, delete, and extract modules in your library of program files.

MS-LIB prompts you for input and output file and module names.

MS-LIB can be run with an automatic response file to answer the library prompts.

MS-LIB produces a cross-reference of symbols in the library modules.

Microsoft CREF Cross-Reference Utility

MS-CREF produces a cross-reference listing of all symbolic names in the Macro Assembler source program, giving both the source line number of the definition and the source line numbers of all other references to the symbols.

Microsoft DEBUG Utility

DEBUG provides a controlled testing environment for binary and executable object files.

DEBUG eliminates the need to reassemble a program to see if a problem has been fixed by a minor change.

DEBUG allows you to alter the contents of a file or the contents of a CPU register, and then immediately reexecute a program to check on the validity of the changes.

Using These Manuals

These manuals are designed to be used as a set and individually. Each manual is mostly self-contained and refers to the other manuals only at junctures in the software. The overview given below describes the flow of program development from creating a source file through program execution. The processes described in this overview are echoed and expanded in overviews in each of the manuals contained in the Microsoft Macro Assembler Manual.

Also, note that each manual has its own index.

Figure 1 illustrates an overview of the Microsoft Macro Assembler Manual.

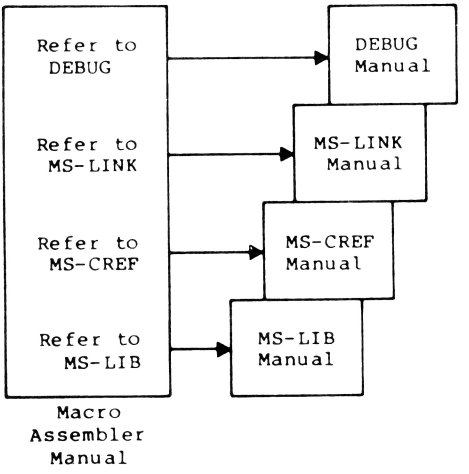


Figure 1. Overview, Macro Assembler Manual

Each of these manuals is used independently. References between manuals reflect junctures in the software.

Syntax Notation

The following notation is used throughout this manual in descriptions of command and statement syntax:

[ ] Square brackets indicate that the enclosed entry is optional.

< > Angle brackets indicate data you must enter. When the angle brackets enclose lower case text, you must type in an entry defined by the text; for example, <filename>. When the angle brackets enclose upper case text, you must press the key named by the text; for example, <RETURN>.

{ } Braces indicate that you have a choice between two or more entries. At least one of the entries enclosed in braces must be chosen unless the entries are also enclosed in square brackets.

... Ellipses indicate that an entry may be repeated as many times as needed or desired.

CAPS Capital letters indicate portions of statements or commands that must be entered, exactly as shown.

All other punctuation, such as commas, colons, slash marks, and equal signs, must be entered exactly as shown.

Figure 2 illustrates the syntax notation used in this manual.

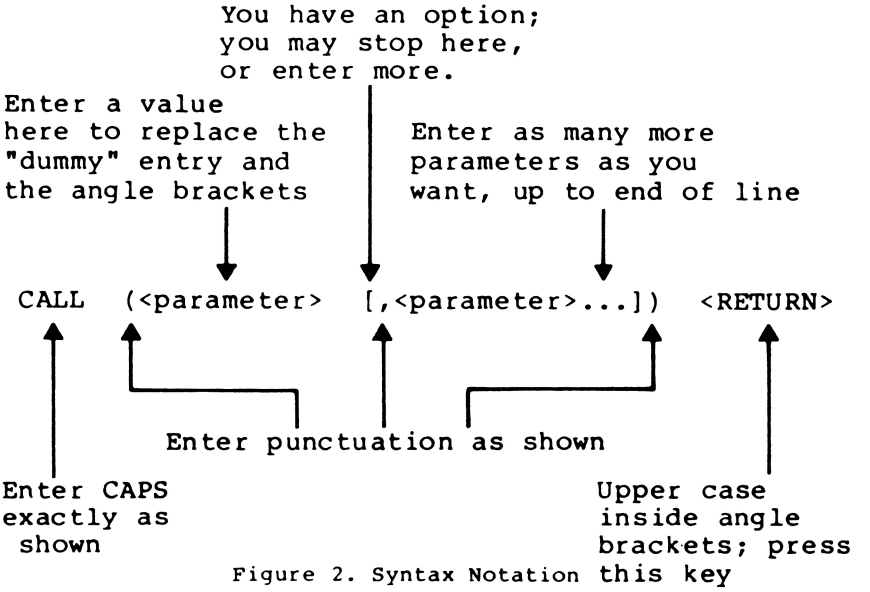


Figure 2. Syntax Notation

Learning More about Assembly Language Programming

These manuals explain how to use MS-DOS utilities and features, but they do not teach you how to program in assembly language.

We assume that you have had some experience programming in assembly language. If you do not have any experience, we suggest two courses:

1. Gain some experience on a less sophisticated assembler.

2. Refer to any or all of the following books for assistance:

Morse, Stephen P. The 8086 Primer. Rochelle Park, NJ: Hayden Publishing Co., 1980.

Rector, Russell and George Alexy. The 8086 Book. Berkeley, CA: Osbourne/McGraw-Hill, 1980.

The 8086 Family User's Manual. Santa Clara, CA: Intel Corporation, 1979.

8086/8087/8088 Macro Assembly Language Reference Manual. Santa Clara, CA: Intel Corporation, 1980.

NOTE

Some of the information in these books was based on preliminary data and may not reflect the final functional state of the microprocessors. Information in your Microsoft manuals was based on Microsoft's development of its 16-bit software for the 8086 and 8088.

Overview of Program Development

This overview describes generally the steps of program development. Each step is described fully in the individual product manuals. The numbers in the descriptions match the numbers in the facing diagram.

1. Use EDLIN (the editor in Microsoft's MS-DOS), or other MS-DOS editor, to create an 8086 assembly language source file. Give the source file the filename extension .ASM (Macro Assembler recognizes .ASM as the default).

2. Assemble the source file with Macro Assembler, which outputs an assembled object file with the default filename extension .OBJ (2a). Assembled files, your program files (2b), can be linked together in step 3.

   Macro Assembler (optionally) creates two types of listing file:

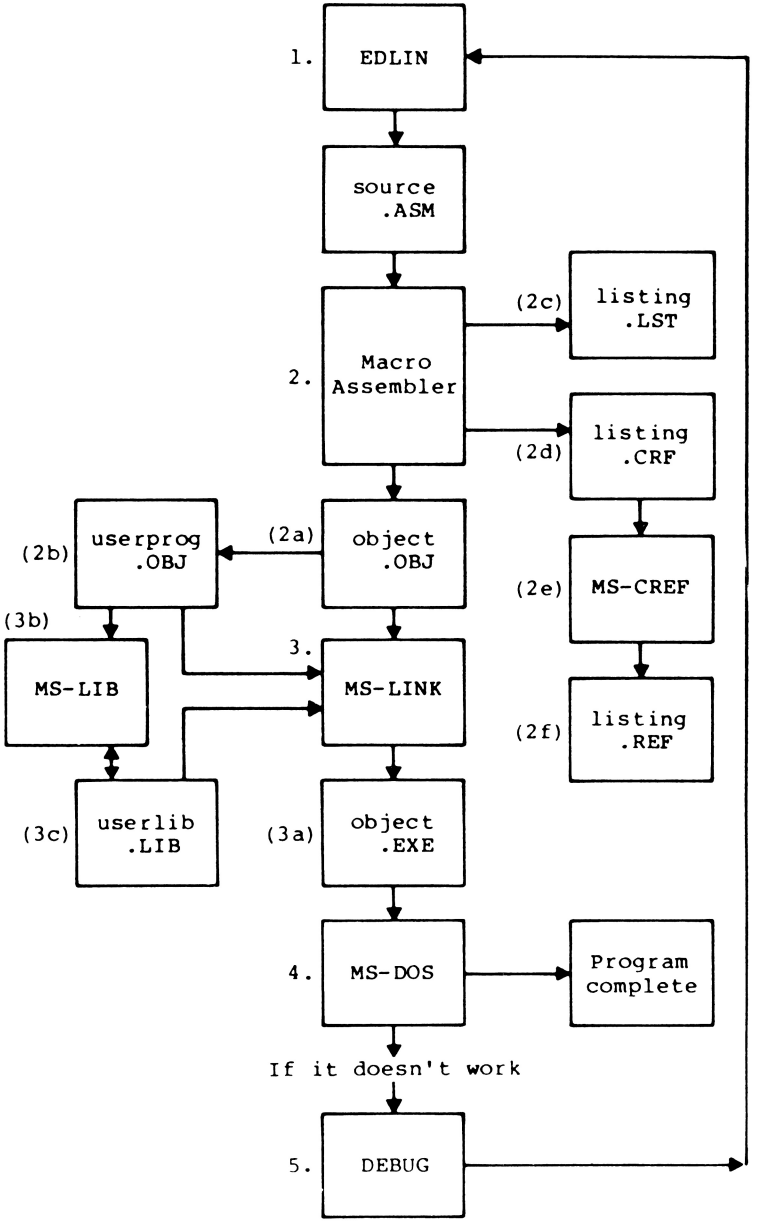
   (2c) a normal listing file which shows assembled code with relative addresses, source statements, and full symbol table;

   (2d) a cross-reference file, a special file with special control characters that allow MS-CREF (2e) to create a list showing the source line number of every symbol's definition and all references to it (2f). When a cross-reference file is created, the normal listing file (with the .LST extension) has line numbers placed into it as references for line numbers following symbols in the cross-reference listing.

3. Link one or more .OBJ modules together, using MS-LINK, to produce an executable object file with the default filename extension .EXE (3a).

   While developing your program, you may want to create a library file for MS-LINK to search to resolve external references. Use MS-LIB (3b) to create user library file(s) (3c) from existing library files (3c) and/or user program object files (2b).

4. Run your assembled and linked program, the .EXE file (3a), under MS-DOS (4). If your program does not run properly, use the DEBUG utility to locate any errors.


1.
EDLIN
source
.ASM
(2c)
listing
.LST
2.
Macro
Assembler
(2d)
listing
.CRF
(2b)
userprog
.OBJ
(2a)
object
.OBJ
(2e)
MS-CREF
(3b)
3.
MS-LIB
MS-LINK
(2f)
listing
.REF
(3c)
userlib
.LIB
(3a)
object
.EXE
4.
MS-DOS
Program
complete
If it doesn't work
5.
DEBUG

Microsoft® Link

Linker Utility

for 8086 and 8088 Microprocessors

Microsoft Corporation

System Requirements

The Microsoft Linker Utility requires:

50K bytes of memory minimum:
- 40K bytes for code and data
- 10K bytes for run space

Disk drive(s):

1 disk drive if and only if output is sent to the same physical disk from which the input was taken. MS-LINK does not allow time to swap disks during operation on a one-drive configuration. Therefore, two disk drives is a more practical configuration.

Contents

Chapter 1 INTRODUCTION

1.1 Overview of MS-LINK Operation 1-2
1.2 Definitions 1-4
1.3 Files That MS-LINK Uses 1-8
1.3.1 Input File Extensions 1-8
1.3.2 Output File Extensions 1-8
1.3.3 VM.TMP (Temporary) File 1-9

Chapter 2 RUNNING MS-LINK

2.1 Method 1: Prompts 2-2
2.2 Method 2: Command Line 2-3
2.3 Method 3: Response File 2-4
2.4 Command Characters 2-6
2.5 MS-LINK Switches 2-8

Chapter 3 MS-LINK TECHNICAL INFORMATION

3.1 Segment Addresses 3-1
3.2 How MS-LINK Assigns Addresses 3-1
3.3 How MS-LINK Combines and Arranges Segments 3-2
3.4 Relocation Fixups 3-5
3.4.1 Short References 3-5
3.4.2 Near Self-Relative References 3-5
3.4.3 Near Segment-Relative References 3-6
3.4.4 Long References 3-6
3.5 Sample MS-LINK Session 3-7
3.6 Error Messages 3-9

Index

CHAPTER 1

INTRODUCTION

The Microsoft(r) Linker Utility (MS(tm)-LINK) is a relocatable linker designed to link separately produced modules of 8086 object code. The input to MS-LINK is a subset of the Intel(tm) object module format standard.

MS-LINK prompts you for all MS-LINK commands. Your answers to these prompts are the commands for MS-LINK.

The output file from MS-LINK (a Run file) is not bound to specific memory addresses, and therefore can be loaded and executed by the operating system at any convenient address. system.

MS-LINK uses a dictionary-indexed library search method, which substantially reduces link time for sessions involving library searches.

MS-LINK is able to link files totaling 1 megabyte.

1.1 OVERVIEW OF MS-LINK OPERATION

MS-LINK performs the following steps to combine object modules and produce a Run file:

1. Reads segments in object modules
2. Assigns addresses to segments
3. Assigns public symbol addresses
4. Reads data in segments
5. Reads all relocation references in object modules
6. Resolves references and determines relocation information
7. Outputs a Run file (executable image) and relocation information

As it combines modules, MS-LINK can search multiple library files for definitions of any external references left unresolved.

MS-LINK also produces a List file that shows external references resolved and any error messages.

MS-LINK uses available memory as much as possible. When available memory is exhausted, MS-LINK creates a disk file (VM.TMP) to use as temporary memory.

The following figure illustrates the MS-LINK operation.

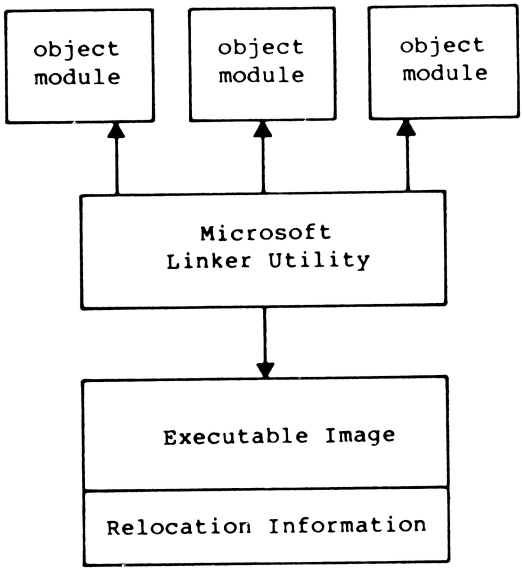


Figure 1. MS-LINK Operation

The executable image contains the concatenated object modules that make the Run file. The relocation information is a list of long references that must change when the executable image is relocated in memory. Refer to Section 3.4.4, "Long References," for more information.

1.2 DEFINITIONS

To help you understand how MS-LINK works, some of the terms used in this section are explained below. Generally, if you are linking object modules compiled from BASIC, Pascal, or a high-level language, you will not need to know these terms. If you are writing and compiling programs in assembly language, however, you will need to understand MS-LINK and the definitions described below.

- Segment

  A segment is a contiguous area of memory up to 64K bytes in length. A segment may be located anywhere in 8086 memory. The contents of a segment are addressed by a canonical frame address and offset within that frame. (Refer to Section 3.1, "Segment Addresses," for further discussion of canonical frames.)

  As seen in Figure 2 below, each segment has a segment name a class name. MS-LINK loads all segments into memory by class name, from the first segment encountered to the last. All segments assigned to the same class are loaded into memory contiguously.

- Group

  A group is a collection of segments that fit within 64K bytes of memory. As seen in Figure 2 below, the segments do not need to be contiguous to form a group. The address of any group is the lowest address of the segments in that group. A program may consist of one or more groups.

  The segments are named to the group by the assembler, by the compiler, or by you. You give the group name in the assembly language program. For the high-level languages (BASIC, FORTRAN, COBOL, Pascal), naming is carried out by the compiler.

  Each group is addressed by a common canonical frame. This frame is the lowest canonical frame of the segments that belong to the group. It is a usual practice in assembler and higher-level languages for the canonical frame address to be contained in a segment register. MS-LINK checks to see that the object modules of a group meet the 64K-byte constraint.

- Class

  A class is a collection of segments. The naming of segments to a class controls the order and relative placement of segments in memory. You give the class name in the assembly language program. For the high-level languages (BASIC, FORTRAN, COBOL, Pascal), the naming is carried out by the compiler. The segments are named to a class at compile time or assembly time.

  The segments of a class are loaded into memory contiguously. The segments are ordered within a class in the order the Linker encounters the segments in the object files. One class precedes another in memory only if a segment for the first class precedes all segments for the second class in the input to MS-LINK. Classes may be loaded across 64K-byte boundaries. Groups may span classes.

- Alignment

  Alignment refers to certain segment boundaries. These can be byte, word, or paragraph boundaries.

  Byte Alignment: A segment can begin on any byte boundary.

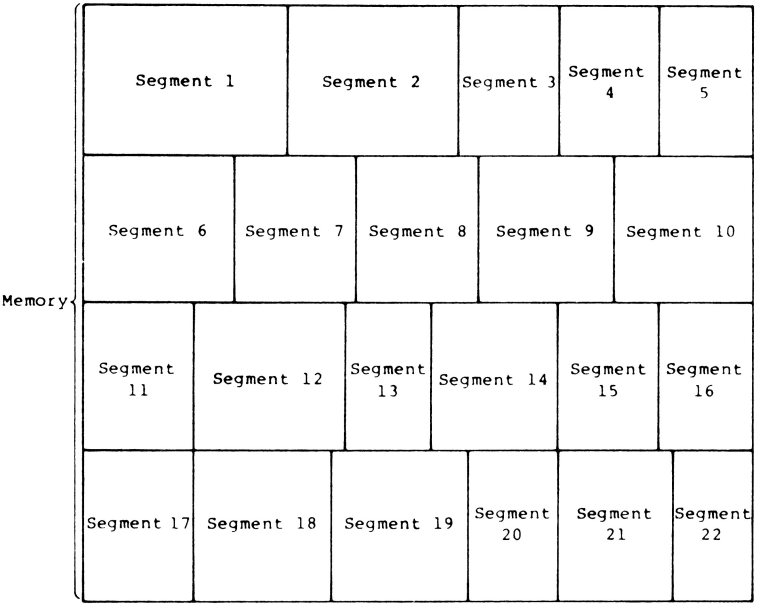
  Word Alignment: The beginning address of a segment must occur on an even address.

  Paragraph Alignment: The beginning address of a segment must occur on a segment (16-byte) boundary.

- Combine Type

  A combine type is an attribute of a segment; it tells the Linker how to combine segments of a like name or it relays other information about the properties of a segment. Combine types are: stack, public, private, and common. The way MS-LINK arranges these combine types is discussed in Section 3.3, "How MS-LINK Combines and Arranges Segments."

Figure 2 illustrates the concepts of segments, classes, and groups.



shaded area = A group (64K bytes addressable)

Figure 2. How Memory Is Divided

Example:

	Segment Name	Segment Class Name
Segment 1	PROG.1	CODE
Segment 2	PROG.2	CODE
Segment 12	PROG.3	DATA

Note that segments 1, 2, and 12 have different segment names but may or may not have the same segment class name. Segments 1, 2, and 12 form a group, with a group address of the lowest address of segment 1 (i.e., the lowest address in memory).

1.3 FILES THAT MS-LINK USES

MS-LINK performs the following tasks:

Works with one or more input files

Produces two output files

May create a temporary disk file

May be directed to search up to eight library files

For each type of file, you can give a three-part file specification. The format of MS-LINK file specifications is the same as that of a disk file:

[d:]<filename>[<.ext>]

where:

d: is the drive designation. Permissible drive designations for MS-LINK are A: through O:. The colon is always required as part of the drive designation.

filename is any legal filename of one to eight characters.

.ext is a one- to three-character extension to the filename. The period is always required as part of the extension.

1.3.1 Input File Extensions

If no filename extensions are given in the input (object) file specifications, MS-LINK will recognize the following extensions by default:

.OBJ	Object
.LIB	Library

1.3.2 Output File Extensions

MS-LINK appends the following default extensions to the output (Run and List) files:

.EXE	Run (may not be overridden)
.MAP	List (may be overridden)

1.3.3 VM.TMP (Temporary) File

MS-LINK uses available memory for the link session. If the files to be linked create an output file that exceeds available memory, MS-LINK will create a temporary file, name it VM.TMP, and put it on the disk in the default drive. If MS-LINK creates VM.TMP, the following message will be displayed:

```
VM.TMP has been created.
Do not change diskette in drive, <d:>
```

Once this message has been displayed, you must not remove the disk from the default drive until the link session ends. If the disk is removed, the operation of MS-LINK will be unpredictable, and MS-LINK might display the error message:

```
Unexpected end of file on VM.TMP
```

The contents of VM.TMP are written to the file named following the Run File: prompt. VM.TMP is a working file only and is deleted at the end of the linking session.

WARNING

Do not use VM.TMP as a filename for any file. If you have a file named VM.TMP on the default drive and MS-LINK needs to create a VM.TMP file, MS-LINK will delete the VM.TMP already on disk and create a new VM.TMP. Thus, the contents of the previous VM.TMP file will be lost.

CHAPTER 2

RUNNING MS-LINK

MS-LINK requires two types of input: a command to start MS-LINK and responses to command prompts. In addition, seven switches control MS-LINK features. Usually, you will type all the commands to MS-LINK on the terminal keyboard. As an option, answers to the command prompts and any switches may be contained in a response file. Command characters can be used to assist you while giving commands to MS-LINK.

MS-LINK can be started in any of three ways. The first method is to type the commands in response to individual prompts. In the second method, you type all commands and switches on the line used to start MS-LINK. To start MS-LINK by the third method, you must create a response file that contains all the necessary commands, and then tell MS-LINK where that file is when you start MS-LINK. The following sections explain these methods in detail.

Summary of Methods to Start MS-LINK

Method 1	LINK
Method 2	LINK <filenames>[/switches]
Method 3	LINK @<filespec>

2.1 METHOD 1: PROMPTS

To start MS-LINK with Method 1, type:

```
LINK
```

MS-LINK will be loaded into memory. MS-LINK will then display four text prompts one at a time. You answer the prompts to command MS-LINK to perform specific tasks.

At the end of each line, you may type one or more switches, preceded by the switch character, a forward slash (/).

The command prompts are summarized below.

PROMPT	RESPONSE
Object Modules [.OBJ]:	List .OBJ files to be linked. They must be separated by blank spaces or plus signs (+). If a plus sign is the last character typed, the prompt will reappear. There is no default; a response is required.
Run File [.EXE]:	Give the filename for executable object code. The default is first-object-filename.EXE. (You cannot change the output extension.)
List File [NUL.MAP]:	Give a filename for the listing. The default is NUL.MAP.
Libraries [.LIB]:	List the filenames to be searched, separated by blank spaces or plus signs (+). If a plus sign is the last character typed, the prompt will reappear. The default is to search for default libraries in the object modules. (Extensions will be changed to .LIB.)

2.2 METHOD 2: COMMAND LINE

To start MS-LINK using Method 2, type all commands on one line. The entries following LINK are responses to the command prompts. The entry fields for the different prompts must be separated by commas. Use the following syntax:

```
LINK <object-list>,<runfile>,<listfile>,<lib-list>[/switch]
```

where: object-list is a list of object modules, separated by plus signs.

runfile is the name of the file that receives the executable output.

listfile is the name of the file that receives the listing.

lib-list is a list of library modules to be searched.

/switch refers to optional switches, which may be placed following any of the response entries (just before any of the commas or after the <lib-list>, as shown).

To select the default for a field, simply type a second comma with no spaces between the two commas.

Example:

```
LINK
FUN+TEXT+TABLE+CARE/P/M,,FUNLIST,COBLIB.LIB
```

This command causes MS-LINK to be loaded; then the object modules FUN.OBJ, TEXT.OBJ, TABLE.OBJ, and CARE.OBJ are loaded. MS-LINK then pauses (as a result of using the /P switch). MS-LINK links the object modules when you press any key, and produces a global symbol map (the /M switch). MS-LINK then defaults to the FUN.EXE run file, creates a list file named FUNLIST.MAP, and searches the library file COBLIB.LIB.

2.3 METHOD 3: RESPONSE FILE

To start MS-LINK with Method 3, type:

```
LINK @<filespec>
```

where: filespec is the name of a response file. A response file contains answers to the MS-LINK prompts (shown in Method 1) and may also contain any of the switches. When naming a response file, use of the filename extension is optional.

Method 3 permits the command that starts MS-LINK to be entered from the keyboard or within a batch file, without requiring you to make any further responses.

To use this option, you must create a response file containing several lines of text, each of which is the response to an MS-LINK prompt. The responses must be in the same order as the MS-LINK prompts discussed in Method 1. If desired, a long response to the Object Modules: or Libraries: prompt may be typed on several lines by using a plus sign (+) to continue the same response onto the next line.

Switches and command characters can be used in the response file the same way they are used for responses typed on the terminal keyboard.

When the MS-LINK session begins, each prompt will be displayed in order with the responses from the response file. If the response file does not contain answers for all the prompts (in the form of filenames, the semicolon command character, or carriage returns), MS-LINK will display the prompt which does not have a response, then wait for you to type a legal response. When a legal response has been typed, MS-LINK continues the link session.

Example:

```
FUN TEXT TABLE CARE
/PAUSE/MAP
FUNLIST
COBLIB.LIB
```

This response file tells MS-LINK to load the four object modules named FUN, TEXT, TABLE, and CARE. MS-LINK pauses to permit you to swap disks before producing a public symbol map. (For a sample public symbol map, see the example session in Section 3.5 of this manual.) Refer to the discussion under /PAUSE in Section 2.5, "MS-LINK Switches," before using this feature. When you press any key, the output files will be named FUN.EXE and FUNLIST.MAP. MS-LINK will then search the library file COBLIB.LIB, and will use default settings for the switches.

2.4 COMMAND CHARACTERS

MS-LINK recognizes the following three command characters:

Plus sign — Use the plus sign (+) to separate entries and to extend the current line in response to the Object Modules: and Libraries: prompts. (A blank space may be used to separate object modules.) To type a large number of responses (each may be very long), type a plus sign then press <RETURN> at the end of the line to extend it. If this is the last entry following these two prompts, MS-LINK will prompt you for more module names. When the Object Modules: or Libraries: prompt appears again, continue to type responses. When all the modules to be linked and libraries to be searched have been listed, be sure the response line ends with a module name and a <RETURN> and not a plus sign and <RETURN>.

Example:

```
Object Modules [.OBJ]:
FUN TEXT TABLE CARE+<RETURN>
Object Modules [.OBJ]:
FOO+FLIPFLOP+JUNQUE+<RETURN>
Object Modules [.OBJ]: CORSAIR<RETURN>
```

Semicolon — To select default responses to the remaining prompts, use a single semicolon (;) followed immediately by a carriage return at any time after the first prompt (Run File:). This feature saves time and overrides the need to press a series of <RETURN> keys.

NOTE

Once the semicolon has been entered (by pressing the <RETURN> key), you can no longer respond to any of the prompts for that link session. Therefore, do not use the semicolon to skip prompts. To skip prompts, use the <RETURN> key.

Example:

```
Object Modules [.OBJ]: FUN TEXT TABLE CARE
<RETURN>
Run Module [FUN.EXE]: _;<RETURN>
```

No other prompts will appear, and MS-LINK will use the default values (including FUN.MAP for the list file).

<CONTROL-C> Use the <CONTROL-C> key to abort the link session at any time. If you type an erroneous response, such as the wrong filename or an incorrectly spelled filename, you must press <CONTROL-C> to exit MS-LINK, then restart MS-LINK. If the error has been typed but you have not pressed the <RETURN> key, you may delete the erroneous characters with the backspace key, for that line only.

2.5 MS-LINK SWITCHES

The seven MS-LINK switches control various MS-LINK functions. Switches must be typed at the end of a prompt response, regardless of which method is used to start MS-LINK. Switches may be grouped at the end of any response, or may be scattered at the end of several responses. If more than one switch is typed at the end of a response, each switch must be preceded by a forward slash (/).

All switches may be abbreviated. The only restriction is that an abbreviation must be sequential from the first letter through the last typed; no gaps or transpositions are allowed. For example, the following is a list of legal and illegal abbreviations for the /DSALLOCATE switch:

Legal	Illegal
/D	/DSL
/DS	/DAL
/DSA	/DLC
/DSALLOCA	/DSALLOCT

/DSALLOCATE

Using the /DSALLOCATE switch tells MS-LINK to load all data at the high end of the Data Segment. Otherwise, MS-LINK loads all data at the low end of the Data Segment. At runtime, the DS pointer is set to the lowest possible address to allow the entire DS segment to be used. Use of the /DSALLOCATE switch in combination with the default load low (that is, the /HIGH switch is not used) permits the user application to dynamically allocate any available memory below the area specifically allocated within DGroup, yet remain addressable by the same DS pointer. This dynamic allocation is needed for Pascal and FORTRAN programs.

NOTE

Your application program may dynamically allocate up to 64K bytes (or the actual amount of memory available) less the amount allocated within DGroup.

/HIGH

Use of the /HIGH switch causes MS-LINK to place the run file as high as possible in memory. Otherwise, MS-LINK places the run file as low as possible.

IMPORTANT

Do not use the /HIGH switch with Pascal or FORTRAN programs.

/LINENUMBERS

The /LINENUMBERS switch tells MS-LINK to include in the list file the line numbers and addresses of the source statements in the input modules. Otherwise, line numbers are not included in the list file.

NOTE

Some compilers produce object modules that do not contain line number information. In these cases, of course, MS-LINK cannot include line numbers.

/MAP

/MAP directs MS-LINK to list all public (global) symbols defined in the input modules. If /MAP is not given, MS-LINK will list only errors (including undefined globals).

The symbols are listed alphabetically at the end of the list file. For each symbol, MS-LINK lists its value and its segment:offset location in the Run file.

/PAUSE

The /PAUSE switch causes MS-LINK to pause in the link session when the switch is encountered. Normally, MS-LINK performs the linking session from beginning to end without stopping. This switch allows you to swap disks before MS-LINK outputs the Run (.EXE) file.

When MS-LINK encounters the /PAUSE switch, it displays the message:

```
About to generate .EXE file
Change disks <hit any key>
```

MS-LINK resumes processing when you press any key.

CAUTION

Do not remove the disk that will receive the list file, or the disk used for the VM.TMP file, if one has been created.

/STACK:<number>

number represents any positive numeric value (in hexadecimal radix) up to 65536 bytes. If a value from 1 to 511 is typed, MS-LINK will use 512. If the /STACK switch is not used for a link session, MS-LINK will calculate the necessary stack size automatically.

All compilers and assemblers should provide information in the object modules that allow the Linker to compute the required stack size.

At least one object (input) module must contain a stack allocation statement. If not, MS-LINK will display the following error message:

```
WARNING: NO STACK STATEMENT
```

/NO

/NO is short for NODEFAULTLIBRARYSEARCH. This switch tells MS-LINK not to search the default (product) libraries in the object modules. For example, if you are linking object modules in Pascal, specifying the /NO switch tells MS-LINK not to automatically search the library named PASCAL.LIB to resolve external references.

CHAPTER 3

MS-LINK TECHNICAL INFORMATION

3.1 SEGMENT ADDRESSES

The 8086 must be able to address all segments in memory. Any 20-bit number can be addressed. The 8086 represents these numbers as two 16-bit numbers; for example, HEX F:12. The F represents a canonical frame address and the 12 is the offset. The canonical frame address is the largest frame address or segment address that can contain the segment. An offset is the segment's location, offset from the beginning of the canonical frame.

The Linker recognizes a segment by its canonical frame address and its offset within the frame.

To convert the segmented address F:12 to a 20-bit number, shift the frame address left 4 bits, and add the offset. Using the above example:

```
      F0
+     12
     ---

F:12 = 102   (20-bit address)
```

3.2 HOW MS-LINK ASSIGNS ADDRESSES

To assign addresses to segments, MS-LINK:

1. Orders each segment by segment and class name.

2. On the basis of the alignment and size of each segment (assuming they are contiguous), the Linker assigns a frame address and an offset to each

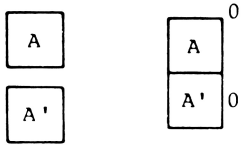
segment. This information is used for resolving relocatable references. The addresses start at 0:0.

3.3 HOW MS-LINK COMBINES AND ARRANGES SEGMENTS

MS-LINK works with four combine types, which are declared in the source module for the assembler or compiler: private, public, stack, and common. The memory combine type available in Microsoft's Macro Assembler is processed the same as public combine type. MS-LINK does not automatically place memory combine type as the highest segments (as defined in the Intel standard).

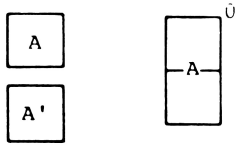
MS-LINK arranges these combine types as follows:

Private

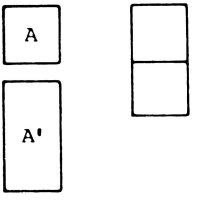


Private segments are loaded separately and remain separate. They may be physically (but not logically) contiguous even if the segments have the same name. Each private segment has its own canonical frame.

Public and Stack



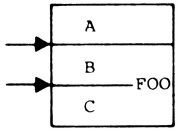
Public and stack segments of the same name and class name are loaded contiguously. Offset is from the beginning of the first segment loaded through the last segment loaded. There is only one canonical frame for all public segments of the same name and class name. Stack and memory combine types are treated the same as public combine types. However, the Stack Pointer is set to the last address of the first stack segment.



Common segments of the same name and class name are loaded overlapping one another. There is only one canonical frame for all common segments of the same name. The length of the common area is the length of the longest segment.

Placing segments in a group in the assembler provides offset addressing of items from a single canonical frame for all segments in that group.

DS:DGROUP--->XXXX0H 0 -- relative offset



Any number of other segments may intervene between segments of a group. Thus, the offset of FOO may be greater than the size of segments in the group combined, but no larger than 64K.

An operand of DGROUP:FOO in assembly language returns the offset of FOO from the beginning of the first segment (segment A here).

Segments are partitioned by declared class names. The Linker loads all the segments belonging to the first class name encountered, then loads all the segments of the next class name encountered, and so on until all classes have been loaded.

If your program contains:

```
A   SEGMENT 'FOO'
B   SEGMENT 'BAZ'
C   SEGMENT 'BAZ'
D   SEGMENT 'ZOO'
E   SEGMENT 'FOO'
```

They will be loaded as:

```
'FOO'
   A
   E
'BAZ'
   B
   C
'ZOO'
   D
```

If you are writing assembly language programs, you can control the order of classes in memory by writing a dummy module and listing it first after the MS-LINK Object Modules: prompt. The dummy module declares segments into classes in the order you want the classes loaded.

WARNING

> Do not use this method with BASIC, COBOL, FORTRAN, or Pascal programs. Allow the compiler and the Linker to perform their tasks in the normal way.

Example:

```
A       SEGMENT 'CODE'
A       ENDS
B       SEGMENT 'CONST'
B       ENDS
C       SEGMENT 'DATA'
C       ENDS
D       SEGMENT STACK   'STACK'
D       ENDS
E       SEGMENT 'MEMORY'
E       ENDS
```

Make sure you declare all classes to be used in your program in this module. If you do not, you lose absolute control over the ordering of classes.

Also, if you want memory combine type to be loaded as the last segments of your program, you can use this method. Simply add MEMORY between SEGMENT and 'MEMORY' in the E segment line above. Note, however, that these segments are loaded last only because you imposed this control on them, not because of any inherent capability in the Linker or assembler operations.

3.4 RELOCATION FIXUPS

MS-LINK performs relocation fixups (i.e., resolves) on four types of references in object modules:

Short

Near Self-Relative

Near Segment-Relative

Long

These references and the Linker's fixups are described in the next sections.

3.4.1 Short References

Short references are all self-relative. The implication is that the frame address of the target and source frames are the same. MS-LINK will generate the fixup error message

Fixup offset exceeds field width

under the following conditions:

1. The target and source frame addresses are different.
2. The target is more than 128 bytes before or after the source frame address.

The resulting value of the short reference must fit into 1 signed byte.

3.4.2 Near Self-Relative References

When near self-relative references are used, the frame address of the target and source frames are the same. MS-LINK will generate the fixup error message under the following conditions:

1. The target and source frame addresses are different.

2. The target is more than 32K before or after the source frame address.

The resulting value of the near self-relative reference must fit into one signed word (16 bits).

3.4.3 Near Segment-Relative References

Given the target's canonical frame, another frame is specified (via an ASSUME directive or the : operator in assembly language; or via a high-level language convention). The target must be addressable through the canonical frame specified. MS-LINK will generate the fixup error message under the following conditions:

1. The offset of the target within the specified frame is greater than 64K or less than 0.

2. The beginning of the canonical frame of the target is not addressable by the specified frame.

The resulting value of a near segment-relative reference must be an unsigned 16-bit quantity.

3.4.4 Long References

Long references have a target frame and another frame (specified by an ASSUME or by a high-level language). The target must be addressable through the canonical frame specified. MS-LINK will generate the fixup error message under the following conditions:

1. The offset of the target within the specified frame is greater than 64K or less than zero.

2. The beginning of the canonical frame of the target is not addressable by the specified frame.

The resulting value of a long reference must be a frame address and an offset.

3.5 SAMPLE MS-LINK SESSION

The following example illustrates the type of information that is displayed during an MS-LINK session.

In response to the MS-DOS prompt (>), the system responds with the following messages and prompts. Answers to the prompts are underlined. Note that pathnames are supported under MS-DOS 2.0. Therefore, your answers to MS-LINK prompts can be full pathnames instead of filenames.

```
Microsoft Object Linker V.2.00
(C) Copyright 1982 by Microsoft Inc.

Object Modules [.OBJ]: IO SYSINIT
Run File [IO.EXE]:
List File [NUL.MAP]: IO /MAP
Libraries [.LIB]: ;
```

Notes:

1. By specifying /MAP, you can get both a sorted alphabetic listing and a sorted address listing of public symbols.

2. By responding PRN to the List File: prompt, you can redirect your output to the printer.

3. By specifying the /LINE switch, MS-LINK gives you a listing of all line numbers for all modules. (Note that the /LINE switch can generate a large volume of output.)

4. By pressing <RETURN> in response to the Libraries: prompt, an automatic library search is performed.

Once MS-LINK locates all libraries, the Linker map displays a list of segments in the order of their appearance within the load module. The list might look like this:

```
Start     Stop      Length     Name
00000H    009ECH    09EDH      CODE
009F0H    01166H    0777H      SYSINITSEG
```

The information in the Start and Stop columns shows the 20-bit hex address of each segment relative to location zero. Location zero is the beginning of the load module.

Because the /MAP switch was used, MS-LINK displays the public symbols by name and value. For example:

```
ADDRESS         PUBLICS BY NAME
009F:0012       BUFFERS
009F:0005       CURRENT_DOS_LOCATION
009F:0011       DEFAULT_DRIVE
009F:000B       DEVICE_LIST
009F:0013       FILES
009F:0009       FINAL_DOS_LOCATION
009F:000F       MEMORY_SIZE
009F:0000       SYSINIT

ADDRESS         PUBLICS BY VALUE
009F:0000       SYSINIT
009F:0005       CURRENT_DOS_LOCATION
009F:0009       FINAL_DOS_LOCATION
009F:000B       DEVICE_LIST
009F:000F       MEMORY_SIZE
009F:0011       DEFAULT_DRIVE
009F:0012       BUFFERS
009F:0013       FILES
```

The addresses of the public symbols are in the frame:offset format, showing the location relative to zero as the beginning of the load module. In some cases, an entry may look like this:

```
780:A2
```

This entry appears to be the address of a load module that is almost one megabyte in size. Actually, the area being referenced is relative to a segment base that is pointing to a segment below the relative zero beginning of the load module. This condition produces a pointer that has effectively gone negative.

When MS-LINK has completed processing, the following message is displayed:

```
Program entry point at 0009F:0000
```

3.6 ERROR MESSAGES

All messages, except for the warning messages, cause the MS-LINK session to end. After you locate and correct a problem, you must rerun MS-LINK.

Messages appear in the List file and are displayed on the screen. If you direct the List file to CON, the error messages will not be displayed on the screen.

Attempt to access data outside of segment bounds, possibly bad object module
: There is probably a bad object file.

Bad numeric parameter
: Numeric value is not in digits.

Cannot open temporary file
: MS-LINK is unable to create the file VM.TMP because the disk directory is full. Insert a new disk. Do not remove the disk that will receive the List.MAP file.

Error: dup record too complex
: The DUP record in the assembly language module is too complex. Simplify the DUP record in your assembly language program.

Error: fixup offset exceeds field width
: An assembly language instruction refers to an address with a short instruction instead of a long instruction. Edit your assembly language source and reassemble.

Input file read error
: There is probably a bad object file.

Invalid object module
: An object module(s) is incorrectly formed or incomplete (as when assembly is stopped in the middle).

Symbol defined more than once
: MS-LINK found two or more modules that define a single symbol name.

Program size or number of segments exceeds capacity of linker
: The total size may not exceed 384K bytes, and the number of segments may not exceed 255.

Requested stack size exceeds 64k
: Specify a size greater than or equal to 64K bytes with the /STACK switch.

Segment size exceeds 64k
: 64K bytes is the addressing system limit.

Symbol table capacity exceeded
: Too many and/or very long names were typed, exceeding the limit of approximately 25K bytes.

Too many external symbols in one module
: The limit is 256 external symbols per module.

Too many groups
: The limit is 10 groups.

Too many libraries specified
: The limit is 8 libraries.

Too many public symbols
: The limit is 1024 public symbols.

Too many segments or classes
: The limit is 256 (segments and classes taken together).

Unresolved externals: <list>
: The external symbols listed have no defining module among the modules or library files specified.

VM read error
This is a disk error; it is not caused by MS-LINK.

Warning: no stack segment
None of the object modules specified contains a statement allocating stack space, although the /STACK switch was specified.

Warning: segment of absolute or unknown type
There is a bad object module, or an attempt has been made to link modules that MS-LINK cannot handle (e.g., an absolute object module).

Write error in TMP file
No more disk space remains to expand the VM.TMP file.

Write error on run file
Usually, there is not enough disk space for the run file.

INDEX

+ (command character) (MS-LINK) 2-6
.EXE (MS-LINK) 1-8
.LIB (MS-LINK) 1-8
.MAP (MS-LINK) 1-8
.OBJ (MS-LINK) 1-8
; (command character) (MS-LINK) 2-6

Alignment 1-5

Canonical frame address . . . 3-1
Class 1-5
Class names 3-3
Combine type 1-5
Command Characters
  + (MS-LINK) 2-6
  ; (MS-LINK) 2-6
Command Characters (MS-LINK) 2-6
Command Prompts
  Libraries (MS-LINK) . . . 2-2
  List File (MS-LINK) . . . 2-2
  Object Modules (MS-LINK) . 2-2
  Run File (MS-LINK) 2-2
  Summary of (MS-LINK) . . . 2-2

Drive designations (MS-LINK) 1-8

Error messages 3-9
Executable image . . . . . 1-2 to 1-3

Filename extensions - default
  .EXE (MS-LINK) 1-8
  .MAP (MS-LINK) 1-8
  .OBJ (MS-LINK) 1-8
Filename extensions - default (MS-LINK) 1-8
Files that MS-LINK uses (MS-LINK) 1-8
Fixup error 3-5

Group 1-4

How MS-LINK combines and arranges segments 3-2

Library files 1-2
List file 1-2
Long references 1-3, 3-6

Near segment-relative references 3-6
Near self-relative references 3-5

Offset 3-1
Offset addressing 3-3
Overviews
  MS-LINK operation 1-2

Pathnames 3-7
Public symbols 3-8

Relocation fixups 3-5
  long 3-5
  near segment-relative . . . 3-5
  near self-relative 3-5
  short 3-5
Response file (MS-LINK) . . 2-4
Run file . . . . . 1-1 to 1-3

Sample MS-LINK session 3-7
Segment 1-4
Segment addresses 3-1
Short references 3-5
Starting
  Summary of Methods (MS-LINK) 2-1
Switches
  MS-LINK
    /DSALLOCATE (MS-LINK) . 2-8
    /HIGH (MS-LINK) 2-9
    /LINENUMBERS (MS-LINK) . 2-9
    /MAP (MS-LINK) 2-9
    /NO (MS-LINK) 2-11
    /PAUSE (MS-LINK) 2-10
    /STACK (MS-LINK) 2-10

VM.TMP 1-2
VM.TMP (MS-LINK) 1-9

Microsoft® LIB

Library Manager

for 8086 and 8088 Microprocessors

Microsoft Corporation

System Requirements

The Microsoft LIB Library Manager requires:

38K bytes of memory minimum:
- 28K bytes for code
- 10K bytes for run space

Disk drive(s):

One disk drive if and only if output is sent to the same physical disk from which the input was taken. The Microsoft LIB Library Manager does not allow time to swap disks during operation on a one-drive configuration. Therefore, two disk drives is a more practical configuration.

Contents

Chapter 1 INTRODUCTION

1.1 Features of MS-LIB 1-1
1.2 Overview of MS-LIB Operation 1-2

Chapter 2 RUNNING MS-LIB

2.1 How to Start MS-LIB 2-1
2.1.1 Method 1: Prompts 2-2
2.1.2 Method 2: Command Line 2-3
2.1.3 Method 3: Response File 2-5
2.2 Command Prompts 2-7
2.3 Command Characters 2-9
+ - append 2-9
- - delete 2-9
* - extract 2-10
; - default remaining prompts 2-10
& - continuation 2-11
CONTROL-C - program abort 2-12

Chapter 3 ERROR MESSAGES

Index

CHAPTER 1

INTRODUCTION

1.1 FEATURES OF MS-LIB

Microsoft LIB is a library manager. With MS-LIB, you can:

Create and modify library files that are used with Microsoft's MS-LINK linker utility

Add object files to a library

Delete modules from a library

Extract modules from a library and place the extracted modules into separate object files

MS-LIB can create either general or special libraries, for a variety of programs or for specific programs. With MS-LIB you can create a library, or you can create a library for one program only. The result is fast linking and more efficient execution for a language compiler or for one program.

You can modify individual modules within a library by extracting the modules, making changes, then adding the modules to the library again. You can also replace an existing module with a different module or with a new version of an existing module.

The command scanner in MS-LIB is also used in Microsoft MS-LINK, MS-Pascal, MS-FORTRAN, and other 16-bit Microsoft products. If you have used any of these products, using MS-LIB should be familiar to you. Command syntax is straightforward, and MS-LIB prompts you for commands that you have not supplied.

1.2 OVERVIEW OF MS-LIB OPERATION

MS-LIB performs five library manager functions:

Deletes modules

Extracts a module and places it in a separate object file

Appends an object file as a module of a library

Replaces a module in the library file with a new module

Creates a library file

During each library session, MS-LIB deletes or extracts modules, then appends new ones to the library file. MS-LIB reads each module into memory, checks it for consistency, and writes it back to the file. If you delete a module, MS-LIB reads that module into memory but does not write it back to the file. When MS-LIB writes back the next module to be retained, it places that module at the end of the last module written. This procedure effectively "closes up" the disk space to keep the library file from growing too large.

When MS-LIB has read the library file, it appends any new modules to the end of the file. Finally, MS-LIB creates the index, which MS-LINK uses to find modules and symbols in the library file. MS-LIB will output a cross-reference listing of the PUBLIC symbols in the library, if you request such a listing.

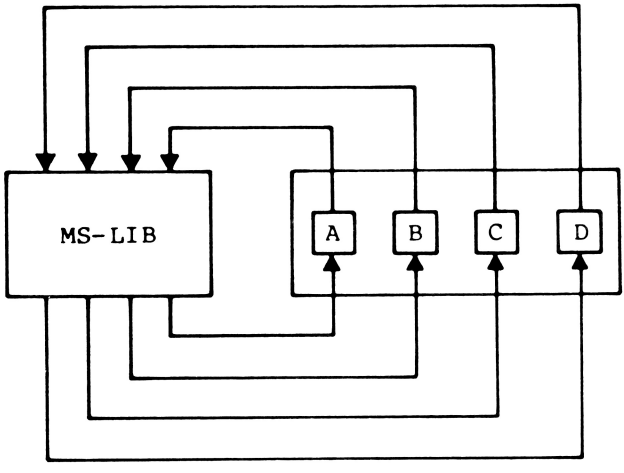
Example:

```
LIBx PASCAL+HEAP-HEAP;
```

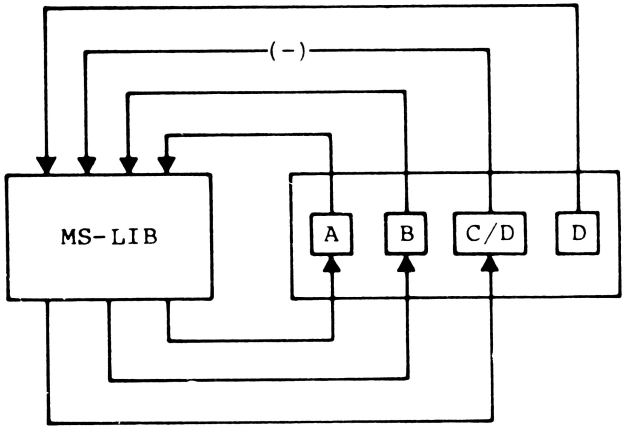
This command first deletes the library module HEAP from the library file, then adds the file HEAP.OBJ as the last module in the library. Note that the replace function is simply the delete-append functions in succession. Also note that you can specify delete, append, or extract functions in any order. This order of execution prevents confusion in MS-LIB when a new version of a module replaces a version in the library file.

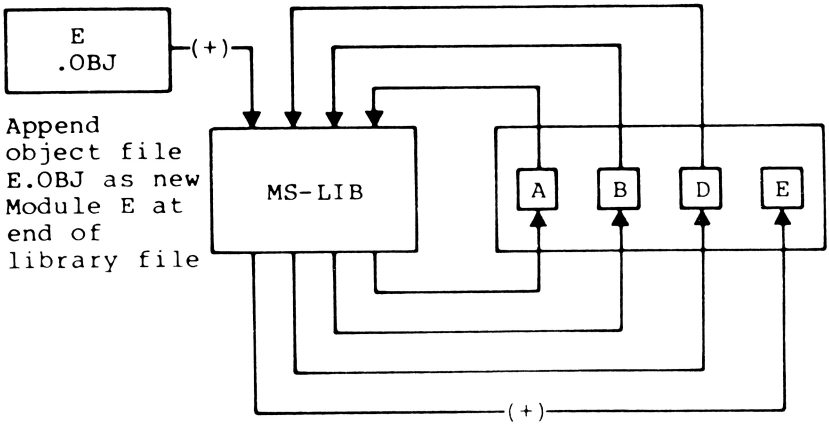
The following figure illustrates the MS-LIB operation.

Consistency check only



Delete Module C; Module D written to space of Module C





Append object file E.OBJ as new Module E at end of library file

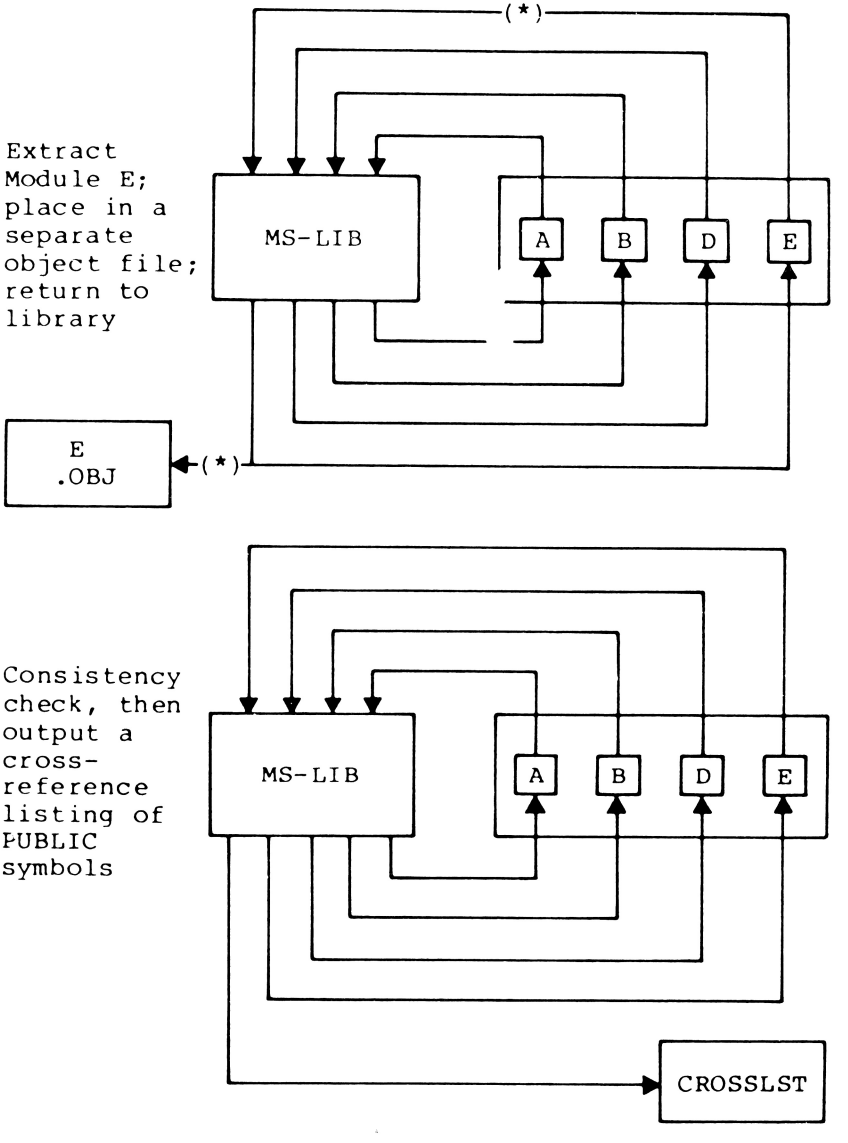


Figure 1. MS-LIB Operation

CHAPTER 2

RUNNING MS-LIB

Running MS-LIB requires two types of commands: a command to start MS-LIB and answers to command prompts. Usually you will type all the commands to MS-LIB on a command line or in response to MS-LIB prompts. As an option, answers to the command prompts may be contained in a response file. Command characters can be used to assist you while giving commands to MS-LIB.

2.1 HOW TO START MS-LIB

There are three ways to start MS-LIB. With the first method, you type the commands as answers to individual prompts. By the second method, you type all commands on the line used to start MS-LIB. As a third option, you can create a response file that contains all the necessary commands.

Summary of Methods to Start MS-LIB

Method 1	LIB
Method 2	LIB <library><operations>,<listing>
Method 3	LIB @<filespec>

2.1.1 Method 1: Prompts

To start MS-LIB with method 1, type:

```
LIB
```

MS-LIB will be loaded into memory. MS-LIB will then display three text prompts that appear one at a time. You answer the prompts, commanding MS-LIB to perform specific tasks.

The command prompts are summarized here and described more fully in the Section 2.2, "Command Prompts."

Summary of Command Prompts

PROMPT	RESPONSES
Library File:	List filename of library to be manipulated. (The default is the filename extension .LIB.)
Operation:	List command character(s) followed by module name(s) or object filename(s). (The default is no changes. The default object filename extension is .OBJ.)
List file:	List filename for a cross-reference listing file. (The default is NUL; i.e., no file.)

NOTE

The distinction between an object file and a module (or object module) is that the file possesses a drive designation (even if it is the default drive) and a filename extension. Object modules possess neither of these.

2.1.2 Method 2: Command Line

Type:

LIB <library><operations>,<listing>

The entries following LIB are responses to the command prompts. The <library> and <operations> fields and all operations entries must be separated by one of the command characters plus, minus, or asterisk (+, -, or *). If a cross-reference listing is wanted, the name of the file must be separated from the last operations entry by a comma.

where: <library> is the name of a library file. MS-LIB assumes that the filename extension is .OBJ, which you may override by specifying a different extension. If the filename given for the <library> field does not exist, MS-LIB will prompt you:

Library file does not exist. Create?

Type Yes to create a new library file. Type No to abort the library session.

<operations> is a command to delete a module, append an object file as a module, or extract a module as an object file from the library file. Use the three command characters plus, minus, and asterisk to direct MS-LIB to append, delete, or extract modules and object files.

<listing> is the name of the file you want to receive the cross-reference listing of PUBLIC symbols in the modules in the library. The list is compiled after all module manipulation has taken place.

If you type a library filename followed immediately by a semicolon, MS-LIB will read through the library file and perform a consistency check. No changes will be made to the modules in the library file.

If you type a library filename followed immediately by a comma and a listing filename, MS-LIB will perform its consistency check of the library file, then produce the cross-reference listing file.

Examples:

```
LIB PASCAL-HEAP+HEAP;
```

This example causes MS-LIB to delete the module HEAP from the library file PASCAL.LIB, then append the object file HEAP.OBJ as the last module of PASCAL.LIB (the module will be named HEAP). The MS-LIB semicolon command character indicates that MS-LIB should use the default responses for the remaining prompts. Refer to Section 2.3, "Command Characters," for more information.

```
LIB PASCAL
```

This example causes MS-LIB to perform a consistency check of the library file PASCAL.LIB. No other action is performed.

```
LIB PASCAL,PASCROSS.PUB
```

This example causes MS-LIB to perform a consistency check of the library file PASCAL.LIB, then output a cross-reference listing file named PASCROSS.PUB.

If you have many operations to perform during a library session, use the ampersand (&) command character to extend the line so that you can type additional object filenames and module names. Be sure to always include one of the command characters for operations (+, -, *) before the name of each module or object filename.

2.1.3 Method 3: Response File

Type:

```
LIB @<filespec>
```

where: <filespec> is the name of a response file. A response file contains answers to the MS-LIB prompts. Method 3 permits you to conduct the MS-LIB session without user responses to the MS-LIB prompts.

IMPORTANT

Before using method 3 to start MS-LIB, you must first create a response file.

A response file has one text line for each prompt. Responses must appear in the same order as the command prompts appear.

Use command characters in the response file the same way you would for responses typed on the keyboard.

When the library session begins, each prompt will be displayed with the responses from the response file. If the response file does not contain answers for all the prompts, MS-LIB will use the default responses. (No changes will be made to the modules currently in the library file, and no cross-reference listing file will be created.)

If you type a library filename followed immediately by a semicolon, MS-LIB will read through the library file and perform a consistency check. No changes will be made to the modules in the library file.

If you type a library filename, a carriage return, a comma, and then a list filename, MS-LIB will perform its consistency check of the library file, then produce the cross-reference listing file.

Example:

```
PASCAL
+CURSOR+HEAP-HEAP*FOIBLES
CROSSLST
```

This response file causes MS-LIB to delete the module HEAP from the PASCAL.LIB library file; extract the module FOIBLES and place it in an object file named FOIBLES.OBJ; then append the object files CURSOR.OBJ and HEAP.OBJ as the last two modules in the library. Then, MS-LIB will create a cross-reference file named CROSSLST.

2.2 COMMAND PROMPTS

You command MS-LIB by typing responses to three text prompts. After you have typed your response to the current prompt, the next appears. When the last prompt has been answered, MS-LIB performs its library management functions without further command. You will see the operating system prompt when MS-LIB has finished the library session successfully. If the library session is unsuccessful, MS-LIB will display the appropriate error message.

MS-LIB prompts you for the name of the library file, the operation(s) you want to perform, and the name you want to give to a cross-reference listing file (if any).

Command Prompts

Library File:

Type the name of the library file that you want to manipulate. MS-LIB assumes that the filename extension is .LIB. You can override this assumption by giving a filename extension when you type the library filename. Because MS-LIB can manage only one library file at a time, only one filename is allowed in response to this prompt. Additional responses, except the semicolon command character, are ignored.

If you type a library filename and follow it immediately with a semicolon command character, MS-LIB will perform a consistency check only, then return to the operating system. Any errors in the file will be displayed.

If the filename you type does not exist, MS-LIB will display the prompt:

```
Library file does not exist. Create?
```

You must type either Yes or No.

Operation:

Type one of the three command characters for manipulating modules (+, -, *); followed immediately (no space) by the module name or the object filename. The plus sign appends an object file as the last module in the library file (see further discussion under the description of plus sign in the next section). The minus sign deletes a module from the library file. The asterisk extracts a module from the library and places it in a separate object file, with the filename taken from the module name and a filename extension .OBJ.

When you have a large number of modules to manipulate (more than can be typed on one line), type an ampersand (&) as the last character on the line. MS-LIB will repeat the Operation: prompt, which permits you to type additional module names and object filenames.

MS-LIB allows you to perform operations on modules and object files in any order you want.

More information about modules is given in the description of each command character.

List file:

If you want a PUBLIC symbols cross-reference list for the modules in the library file, type the name of a file in which you want MS-LIB to place the cross-reference listing. If you do not type a filename, no cross-reference listing is generated.

The response to the List file: prompt is a file specification. You can specify a drive (or device) designation and a filename extension with the filename. The list file is not given a default filename extension. If you want the file to have a filename extension, you must specify it when typing the filename.

The cross-reference listing file contains two lists. The first list is an alphabetical listing of all PUBLIC symbols. Each symbol name is followed by the name of its module. The second list is an alphabetical list of the modules in the library. Under each module name is an alphabetical listing of the PUBLIC symbols in that module.

2.3 COMMAND CHARACTERS

MS-LIB provides six command characters. Three of the command characters are required in response to the Operation: prompt. The other three command characters provide you with helpful commands to MS-LIB.

Plus sign

Use the plus sign (+), followed by an object filename, to append the object file as the last module in the library named in response to the Library File: prompt. When MS-LIB sees the plus sign, it assumes that the filename extension is .OBJ. You may override this assumption by specifying a different filename extension.

MS-LIB strips the drive designation and the extension from the object file specification, leaving only the filename. For example, if the object file to be appended as a module to a library is

```
B:CURSOR.OBJ
```

a response to the Operation: prompt of

```
+B:CURSOR.OBJ
```

will cause MS-LIB to strip off the B: and the .OBJ, leaving only CURSOR. This becomes a module named CURSOR in the library.

Minus sign

Use the minus sign, followed by a module name, to delete a module from the library file. MS-LIB then "closes up" the disk space left empty by the deletion. This cleanup action keeps the library file from growing larger than necessary. Remember that new modules, even replacement modules, are added to the end of the file, not put into space vacated by deleting modules.

Asterisk

Use the asterisk, followed by a module name, to extract the module from the library file and place it into a separate object file. The module will still exist in the library. (The extraction process copies the module to a separate object file.) The module name is used as the filename. MS-LIB adds the default drive designation and the filename extension .OBJ. For example, if the module to be extracted is

```
CURSOR
```

and the current default disk drive is A:, a reponse to the Operation: prompt of

```
*CURSOR
```

causes MS-LIB to extract the module named CURSOR from the library file and make it an object file with the file specification of:

```
A:CURSOR.OBJ
```

The drive designation and filename extension cannot be overridden. You can, however, rename the file, giving a new filename extension; and/or copy the file to a new disk drive, giving a new filename and/or filename extension.

Semicolon

Use a single semicolon (;), followed immediately by a carriage return at any time after responding to the first prompt (i.e., from Library File: on), to select default responses to the remaining prompts. This feature saves time and overrides the need to answer additional prompts.

NOTE

Once the semicolon has been typed, you can no longer respond to any of the prompts for that library session. Therefore, do not use the semicolon to skip over prompts. To skip prompts, use carriage return.

Example:

```
Library file:  FUN
Operation:  +CURSOR;
```

The remaining prompt will not appear, and MS-LIB will use the default value (no cross-reference file).

Ampersand

Use the ampersand to extend the current line. This command character is only used in response to the Operation: prompt. The number of modules you can append is limited only by disk space. The number of modules you can replace or extract is also limited only by disk space. The number of modules you can delete is limited by the number of modules in the library file.

The line length for a response to any prompt is limited to the line length of your system. For a large number of responses to the Operation: prompt, place an ampersand at the end of a line. MS-LIB will display the Operation: prompt again, and then you can type more responses. For example:

```
Library File:  FUN
Operation:  +CURSOR-HEAP+HEAP*FOIBLES&
Operation:  *INIT+ASSUME+RIDE;
```

MS-LIB will delete the module HEAP; extract the modules FOIBLES and INIT (creating two files, FOIBLES.OBJ and INIT.OBJ); then append the object files CURSOR, HEAP, ASSUME, and RIDE. Note that MS-LIB allows you to type your Operation: reponses in any order. You may use the ampersand character as

many times as needed.

CONTROL-C Use <CONTROL-C> to abort the library session at any time. If you type an incorrect response, such as the wrong filename or module name, or an incorrectly spelled filename or module name, you must press <CONTROL-C> to exit MS-LIB; then you must restart MS-LIB. If the error has been typed and you have not pressed the <RETURN> key, you may delete the erroneous characters for that line only.

Summary of Command Characters

Character	Action
+	Appends an object file as the last module
-	Deletes a module from the library
*	Extracts a module and places in an object file
;	Use default responses to remaining prompts
&	Extends current physical line; repeats command prompt
CONTROL-C	Aborts library session

CHAPTER 3

ERROR MESSAGES

The following are MS-LIB error messages:

<symbol> is a multiply defined PUBLIC. Proceed?
Cause: Two modules define the same public symbol. You are asked to confirm the removal of the definition of the old symbol.

Cure: Remove the PUBLIC declaration from one of the object modules and recompile or reassemble. If you respond No, the library will be left in an indeterminate state.

Allocate error on VM.TMP
Cause: Out of disk space

Cannot create extract file
Cause: No room in directory for extract file

Cannot create list file
Cause: No room in directory for library file

Cannot nest response file
Cause: @filespec in response (or indirect) file

MS-LIB cannot open VM.TMP
Cause: There is no room for VM.TMP in disk directory

Cannot write library file
Cause: Out of disk space

Close error on extract file
Cause: Out of disk space

Error: An internal error has occurred
Contact Microsoft Corporation

Fatal Error: Cannot open input file
Cause: You mistyped an object filename

Fatal Error: Module is not in the library
Cause: You tried to delete a module that is not in the library

Input file read error
Cause: Bad object module or faulty disk

Invalid object module/library
Cause: Bad object module and/or library

Library Disk is full
Cause: No more room on disk

Listing file write error
Cause: Out of disk space

No library file specified
Cause: No response to Library File: prompt

Read error on VM.TMP
Cause: Disk not ready for read

Symbol table capacity exceeded
Cause: Too many public symbols (about 30K chars in symbols)

Too many object modules
Cause: More than 500 object modules

Too many public symbols
Cause: 1024 public symbols maximum

Write error on library/extract file
Cause: Out of disk space

Write error on VM.TMP
Cause: Out of disk space

INDEX

& (command character) 2-11, 2-13
* (command character) 2-10, 2-13
+ (command character) 2-9, 2-13
- (command character) 2-9, 2-13
; (command character) 2-10, 2-13

Command Characters 2-9
  & 2-11, 2-13
  * 2-10, 2-13
  + ... 2-9, 2-13
  - 2-9, 2-13
  ; 2-13
  CONTROL-C 2-12
  Control-C 2-13
  Summary of 2-13
Command Prompts 2-7
  Library file 2-2, 2-7
  List file 2-2, 2-8
  Operation 2-2, 2-8
  Summary of 2-2
Consistency check 2-3, 2-5
CONTROL-C (command character) 2-12
Control-C (command character) 2-13
Creating a new library 2-3, 2-7

Error messages 3-1

Library file (command prompt) 2-2, 2-7
List file (command prompt) . . 2-2, 2-8

Method 1 2-2
Method 2 2-3
Method 3 2-5

Operation (command prompt) . . 2-2, 2-8
Overviews
  MS-LIB operation 1-2

Response File 2-5
Running MS-LIB 2-1

Starting
  Method 1 2-2
  Method 2 2-3
  Method 3 2-5
  Summary of Methods 2-1
Starting MS-LIB 2-1
Summary of methods to start . 2-1

Microsoft® CREF

Cross-Reference Utility

for 8086 and 8088 Microprocessors

Microsoft Corporation

System Requirements

The Microsoft CREF Cross-Reference Utility requires:

24K bytes of memory minimum:
- 14K bytes for code
- 10K bytes for run space

Disk drive(s):
- 1 disk drive if and only if output is sent to the same physical disk from which the input was taken. The Microsoft CREF Cross-Reference Utility does not allow time to swap disks during operation on a one-drive configuration. Therefore, two disk drives is a more practical configuration.

Contents

Chapter 1 INTRODUCTION

1.1 Features of MS-CREF 1-1
1.2 Overview of MS-CREF Operation 1-2

Chapter 2 RUNNING MS-CREF

2.1 How to Create a Cross-Reference File 2-1
2.2 How to Start MS-CREF 2-2
2.2.1 Method 1: Prompts 2-3
2.2.2 Method 2: Command Line 2-4
2.3 Command Characters 2-6
2.4 Format of Cross-Reference Listings 2-6
2.4.1 Example of Cross-Reference Listing 2-7

Chapter 3 ERROR MESSAGES

Chapter 4 FORMAT OF MS-CREF COMPATIBLE FILES

4.1 MS-CREF File Processing 4-1
4.2 Format of Source Files 4-2
4.2.1 First Three Bytes 4-2
4.2.2 Control Symbols 4-2

Index

CHAPTER 1

INTRODUCTION

1.1 FEATURES OF MS-CREF

The Microsoft CREF Cross-Reference Utility can help you in debugging your assembly language programs. MS-CREF outputs an alphabetical listing of all the symbols to a special file created by your assembler. With this listing, you can quickly locate all occurrences of any symbol in your source program by line number.

The cross-reference listing produced by MS-CREF gives you symbol locations, speeding your search and allowing faster debugging.

The MS-CREF listing is used with the symbol table produced by your assembler.

The symbol table listing shows the value, type, and length of each symbol. This information is needed to correct erroneous symbol definitions or uses.

1.2 OVERVIEW OF MS-CREF OPERATION

MS-CREF produces a file with cross-references for symbolic names in your program.

First, you must create a cross-reference file with the assembler. Then, MS-CREF converts this cross-reference file (which has the filename extension .CRF) into an alphabetical listing of the symbols in the file. The cross-reference listing file is given the default filename extension .REF.

Beside each symbol in the listing, MS-CREF lists the line numbers where the symbol occurs in the source program. The line numbers are listed in ascending sequence. The line number where the symbol is defined is indicated by a pound sign (#).

Figure 1 illustrates the MS-CREF operation.

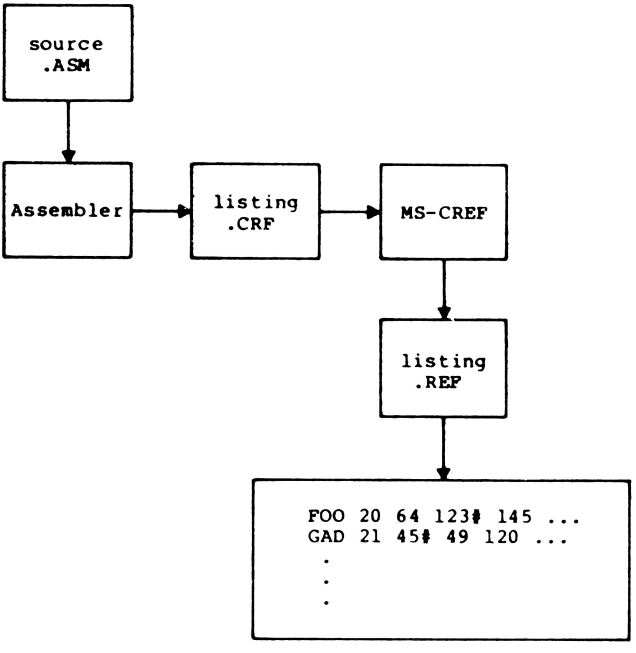


Figure 1. MS-CREF Operation

CHAPTER 2

RUNNING MS-CREF

Running MS-CREF requires two types of commands: a command to start MS-CREF and answers to command prompts. You type all the commands to MS-CREF on a command line or in response to MS-CREF prompts. Command characters can be used to assist you while giving commands to MS-CREF.

Before you can use MS-CREF to create the cross-reference listing, you must first create a cross-reference file using your assembler. This step is described in the next section.

2.1 HOW TO CREATE A CROSS-REFERENCE FILE

A cross-reference file is created during an assembly session. To create a cross-reference file, use the Microsoft Macro Assembler and answer the fourth command prompt with the name of the cross-reference file you want to create.

The fourth assembler prompt is:

```
Cross-reference [NUL.CRF]:
```

If you do not type a filename in response to this prompt, or if you use the default response, the assembler will not create a cross-reference file. Therefore, you must type a filename if you want to create a cross-reference file.

You may also specify which drive or device you want the file saved on, and the filename extension (if different from .CRF). If you assign a filename extension other than .CRF, you must specify the filename extension when naming the file in response to the first MS-CREF prompt. (Refer to Section 2.2, "How to Start MS-CREF," for a description of MS-CREF prompts.)

You are now ready to use MS-CREF to convert the cross-reference file produced by the assembler into a cross-reference listing.

2.2 HOW TO START MS-CREF

MS-CREF may be started two ways. By the first method, you type the commands as answers to individual prompts. By the second method, you type all commands on the line used to start MS-CREF.

Summary of Methods to Start MS-CREF

Method 1	CREF
Method 2	CREF <crffile>,<listing>

2.2.1 Method 1: Prompts

To start MS-CREF using prompts, type:

```
CREF
```

MS-CREF will be loaded into memory. Then, MS-CREF displays two text prompts that appear one at a time. You answer the prompts to command MS-CREF to convert a cross-reference file into a cross-reference listing.

Command Prompts

Cross reference [.CRF]:

Type the name of the cross-reference file you want MS-CREF to convert to a cross-reference listing. The filename is the name you specified when you directed the assembler to produce the cross-reference file.

MS-CREF assumes that the filename extension is .CRF. If you do not specify a filename extension when you type the cross-reference filename, MS-CREF will look for a file with the name you specify and the filename extension .CRF. If your cross-reference file has a different extension, specify that extension when typing the filename.

Refer to Chapter 4, "Format of MS-CREF Compatible Files," for a description of what MS-CREF expects to see in the cross-reference file. You will need this information only if your cross-reference file was not produced by a Microsoft assembler.

Listing [crffile.REF]:

Type the name you want the cross-reference listing file to have. MS-CREF will automatically give the cross-reference listing the filename extension .REF.

If you want you cross-reference listing to have the same filename as the cross-reference file but with the filename extension .REF, simply press the <RETURN> key when the Listing: prompt appears. If you want your cross-reference listing file to be named anything else, or to have any other filename extension, you must type a response following the Listing: prompt.

If you want the listing file placed on a drive or device other than the default drive, specify that drive or device when typing your response to the Listing: prompt.

2.2.2 Method 2: Command Line

To start MS-CREF using the command line, type:

CREF <crffile>,<listing>

MS-CREF will be loaded into memory. Then, MS-CREF converts your cross-reference file into a cross-reference listing.

The entries following CREF are responses to the command prompts. The <crffile> and <listing> fields must be separated by a comma.

where: <crffile> is the name of the cross-reference file produced by your assembler. MS-CREF assumes that the filename extension is .CRF. You may override this default by specifying a different extension. If the file named for the <crffile> does not exist, MS-CREF will display the message:

Fatal I/O Error 110

in File: <crffile>.CRF

MS-CREF will be aborted and the operating system prompt will appear.

<listing> is the name of the file you want to contain the cross-reference listing of symbols in your program.

To select the default filename and extension for the listing file, type a semicolon after the <crffile> name. Refer to the "Command Characters" section for more information on how to use the semicolon.

Examples:

CREF FUN;

This example causes MS-CREF to process the cross-reference file FUN.CRF and to produce a listing file named FUN.REF.

To give the listing file a different filename, extension, or destination, simply specify it when you type the command line.

```
CREF FUN,B:WORK.ARG
```

This example causes MS-CREF to process the cross-reference file named RUN.CRF and to produce a listing file named WORK.ARG, which will be placed on the disk in drive B:.

2.3 COMMAND CHARACTERS

MS-CREF provides two command characters.

Semicolon — Use a single semicolon (;), followed immediately by a carriage return, at any time after responding to the Cross reference: prompt to select the default response to the Listing: prompt. This feature saves time and overrides the need to answer the Listing: prompt.

If you use the semicolon, MS-CREF gives the listing file the filename of the cross-reference file and the default filename extension .REF.

Example:

```
Cross reference [.CRF]:  FUN;
```

MS-CREF will process the cross-reference file named FUN.CRF and output a listing file named FUN.REF.

CONTROL-C — Use <CONTROL-C> at any time to abort the MS-CREF session. If you make a mistake (for example, typing the wrong filename or incorrectly spelling a filename), you must press <CONTROL-C> to exit MS-CREF, and then restart MS-CREF. If the error has been typed but you have not pressed the <RETURN> key, you may delete the erroneous characters, but for that line only.

2.4 FORMAT OF CROSS-REFERENCE LISTINGS

The cross-reference listing is an alphabetical list of all the symbols in your program. Each page begins with the title of the program or program module. Then the symbols are listed. Following each symbol name is a list of the line numbers where the symbol occurs in your program. The line number for the definition has a pound sign (#) appended to it.

An example of a cross-reference listing appears in the next section.

2.4.1 Example Of Cross-Reference Listing

```
MS-CREF      (vers no.)      (date)

ENTX    PASCAL entry for initializing programs<--comes from
                                          TITLE directive

 Symbol Cross-Reference    (# is definition)   Cref-1

AAAXQQ . . . .  37#   38

BEGHQQ . . . .  83    84#  154   176
BEGOQQ . . . .  33   162
BEGXQQ . . . . 113   126#  164   223

CESXQQ . . . .  97    99#  129
CLNEQQ . . . .  67    68#
CODE . . . . .  37   182
CONST. . . . . 104   104   105   110
CRCXQQ . . . .  93    94#  210   215
CRDXQQ . . . .  95    96#  216
CSXEQQ . . . .  65    66#  149
CURHQQ . . . .  85    86#  155

DATA . . . . .  64#   64   100   110
DGROUP . . . . 110#  111   111   111   127   153   171   172
DOSOFF . . . .  98#  198   199
DOSXQQ . . . . 184   204#  219

ENDHQQ . . . .  87    88#  158
ENDOQQ . . . .  33#  195
ENDUQQ . . . .  31#  197
ENDXQQ . . . . 184   194#
ENDYQQ . . . .  32#  196
ENTGQQ . . . .  30#  187
ENTXCM . . . . 182#  183   221

FREXQQ . . . . 169   170#  178

HDRFQQ . . . .  71    72#  151
HDRVQQ . . . .  73    74#  152
HEAP . . . . .  42    44   110
HEAPBEG. . . .  54#  153   172
HEAPLOW. . . .  43   171

INIUQQ . . . .  31   161

MAIN_STARTUP  109#  111   180
MEMORY . . . .  42    48#   48    49   109   110

PNUXQQ . . . .  69    70   150

RECEQQ . . . .  81    82#
```

```
REFEQQ . . . .  77    78#
REPEQQ . . . .  79    80#
RESEQQ . . . .  75    76#  148
ENTX    PASCAL entry for initializing programs

 Symbol Cross-Reference    (# is definition)   Cref-2

SKTOP. . . . .  59#
SMLSTK . . . . 135   137#
STACK. . . . .  53#   53    60   110
STARTMAIN. . . 163   186#  200
STKBQQ . . . .  89    90#  146
STKHQQ . . . .  91    92#  160
```

CHAPTER 3

ERROR MESSAGES

All errors cause MS-CREF to abort. Control is returned to the operating system.

All error messages are displayed in the following format:

```
Fatal I/O Error <error number>
in File:  <filename>
```

where: <filename> is the name of the file where the error occurs.

<error number> is one of the numbers in the following list of errors:

Number	Error
101	Hard data error Unrecoverable disk I/O error
101	Device name error Illegal device specification (for example, X:FOO.CRF)
103	Internal error Report to Microsoft Corporation
104	Internal error Report to Microsoft Corporation
105	Device offline Disk drive door open, no printer attached, or similar device is offline.
106	Internal error Report to Microsoft Corporation
108	Disk full
110	File not found
111	Disk is write protected
112	Internal error Report to Microsoft Corporation
113	Internal error Report to Microsoft Corporation
114	Internal error Report to Microsoft Corporation
115	Internal error Report to Microsoft Corporation

CHAPTER 4

FORMAT OF MS-CREF COMPATIBLE FILES

MS-CREF will process files other than those generated by Microsoft's assembler, as long as the file conforms to the valid MS-CREF format.

4.1 MS-CREF FILE PROCESSING

MS-CREF reads a stream of bytes from the cross-reference file (or source file), sorts them, then emits them as a printable listing file (the .REF file). The symbols are held in memory as a sorted tree. References to the symbols are held in a linked list.

MS-CREF keeps track of line numbers in the source file by the number of end-of-line characters it encounters. Therefore, every line in the source file must contain at least one end-of-line character (see chart below).

MS-CREF places a heading at the top of every page of the listing. The name MS-CREF uses is passed by your assembler from a TITLE (or similar) directive in your source program. The title must be followed by a title symbol (see chart below). If MS-CREF encounters more than one title symbol in the source file, it will use the last title read for all page headings. If MS-CREF does not encounter a title symbol in the file, the title line on the listing will be blank.

4.2 FORMAT OF SOURCE FILES

MS-CREF uses the first three bytes of the source file as format specification data. The rest of the file is processed as a series of records that either begin or end with a byte that identifies the type of record.

4.2.1 First Three Bytes

The PAGE directive in your assembler, which takes arguments for page length and line length, will pass the following information to the cross-reference file:

First Byte
: The number of lines to be printed per page (page length range is from 1 to 255 lines).

Second Byte
: The number of characters per line (line length range is from 1 to 132 characters).

Third Byte
: The Page Symbol (07) that tells MS-CREF that the two preceding bytes define listing page size.

If MS-CREF does not see these first three bytes in the file, it uses default values for page size (page length is 58 lines; line length is 80 characters).

4.2.2 Control Symbols

The two tables below show the types of records that MS-CREF recognizes and the byte values and placement it uses to recognize record types.

Records have a control symbol (which identifies the record type) either as the first byte of the record or as the last byte.

Records That Begin with a Control Symbol

Byte Value*	Control Symbol	Subsequent Bytes
01	Reference symbol	Record is a reference to a symbol name (1 to 80 characters)
02	Define symbol	Record is a definition of a symbol name (1 to 80 characters)
04	End-of-line	(none)
05	End-of-file	1AH

Records That End with a Control Symbol

Byte Value*	Control Symbol	Preceding Bytes
06	Title defined	Record is title text (1 to 80 characters)
07	Page length/ line length	One byte for page length followed by one byte for line length

*For all record types, the byte value represents a control character, as follows:

01 Control-A
02 Control-B
04 Control-D
05 Control-E
06 Control-F
07 Control-G

The Control Symbols are defined as follows:

Reference symbol
: Record contains the name of a symbol that is referenced. The name may be from 1 to 80 ASCII characters long. Additional characters are truncated.

Define symbol
: Record contains the name of a symbol that is defined. The name may be from 1 to 80 ASCII characters long. Additional characters are truncated.

End-of-line
: Record is an end-of-line symbol character only (04H or Control-D).

End-of-file
: Record is the end-of-file character (1AH).

Title defined
: ASCII characters of the title are to be printed at the top of each listing page. The title may be from 1 to 80 characters long. Additional characters are truncated. The last title definition record encountered is used for the title placed at the top of all pages of the listing. If a title definition record is not encountered, the title line on the listing will be left blank.

Page length/line length
: The first byte of the record contains the number of lines to be printed per page (range is from 1 to 255 lines). The second byte contains the number of characters to be printed per line (range is from 1 to 132 characters). The default page length is 58 lines. The default line length is 80 characters.

The following table illustrates CRF file record contents by byte and length of record.

Summary of CRF File Record Contents

Byte Contents	Length of Record
01 symbol_name	2-81 bytes
02 symbol_name	2-81 bytes
04	1 byte
05 lA	2 bytes
title_text 06	2-81 bytes
PL LL 07	3 bytes

INDEX

.CRF (default extension) . . . 1-2
.REF (default extension) . . . 1-2
; (command character) 2-6

Command Characters 2-6
  ; 2-6
  CONTROL-C 2-6
Command Prompts
  Cross-reference [.CRF] . . . 2-3
  Listing [crffile.REF] . . . 2-3
Control symbols 4-2, 4-4
CONTROL-C (command character) 2-6
Creating a cross-reference file 2-1
Cross reference [.CRF] (command prompt) 2-3

Default extensions
  .CRF 1-2
  .REF 1-2

Error messages 3-1

Format of cross-reference listings 2-6
Format of MS-CREF compatible files 4-1

Listing [crffile.REF] (command prompt) 2-3

Method 1 2-3
Method 2 2-4

Overviews
  MS-CREF operation 1-2

Running MS-CREF 2-1

Starting
  Method 1 2-3
  Method 2 2-4
Starting MS-CREF 2-2
Summary of CRF file record contents 4-5
Summary of methods to start . 2-2

Microsoft® DEBUG

Utility

for 8086 and 8088 Microprocessors

Microsoft Corporation

System Requirements

The Microsoft DEBUG Utility requires:

A memory minimum that is program-dependent:
13K bytes for code
Run space is program-dependent

Disk drive(s):
1 disk drive if and only if output is sent to the same physical disk from which the input was taken. Microsoft DEBUG does not allow time to swap disks during operation on a one-drive configuration. Therefore, two disk drives is a more practical configuration.

Contents

Chapter 1 INTRODUCTION

1.1 Overview of DEBUG 1-1
1.2 How to Start DEBUG 1-1
1.2.1 Method 1: DEBUG 1-2
1.2.2 Method 2: Command Line 1-2

Chapter 2 COMMANDS
2.1 Command Information 2-1
2.2 Parameters 2-3
2.3 Error Messages 2-36

Index

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF DEBUG

The Microsoft DEBUG Utility (DEBUG) is a debugging program that provides a controlled testing environment for binary and executable object files. Note that EDLIN is used to alter source files; DEBUG is EDLIN's counterpart for binary files. DEBUG eliminates the need to reassemble a program to see if a problem has been fixed by a minor change. It allows you to alter the contents of a file or the contents of a CPU register, and then to immediately reexecute a program to check on the validity of the changes.

All DEBUG commands may be aborted at any time by pressing <CONTROL-C>. <CONTROL-S> suspends the display, so that you can read it before the output scrolls away. Entering any key other than <CONTROL-C> or <CONTROL-S> restarts the display. All of these commands are consistent with the control character functions available at the MS-DOS command level.

1.2 HOW TO START DEBUG

DEBUG may be started two ways. By the first method, you type all commands in response to the DEBUG prompt (a hyphen). By the second method, you type all commands on the line used to start DEBUG.

Summary of Methods to Start DEBUG

Method 1	DEBUG
Method 2	DEBUG [<filespec> [<arglist>]]

1.2.1 Method 1: DEBUG

To start DEBUG using method 1, type:

```
DEBUG
```

DEBUG responds with the hyphen (-) prompt, signaling that it is ready to accept your commands. Since no filename has been specified, current memory, disk sectors, or disk files can be worked on by using other commands.

Warnings

1. When DEBUG (Version 2.0) is started, it sets up a program header at offset 0 in the program work area. On previous versions of DEBUG, you could overwrite this header. You can still overwrite the default header if no <filespec> is given to DEBUG. If you are debugging a .COM or .EXE file, however, do not tamper with the program header below address 5CH, or DEBUG will terminate.

2. Do not restart a program after the "Program terminated normally" message is displayed. You must reload the program with the N and L commands for it to run properly.

1.2.2 Method 2: Command Line

To start DEBUG using a command line, type:

```
DEBUG [<filespec> [<arglist>]
```

For example, if a <filespec> is specified, then the following is a typical command to start DEBUG:

```
DEBUG FILE.EXE
```

DEBUG then loads FILE.EXE into memory starting at 100 hexadecimal in the lowest available segment. The BX:CX registers are loaded with the number of bytes placed into memory.

An <arglist> may be specified if <filespec> is present. The <arglist> is a list of filename parameters and switches that are to be passed to the program <filespec>. Thus, when <filespec> is loaded into memory, it is loaded as if it had been started with the command:

```
<filespec> <arglist>
```

Here, <filespec> is the file to be debugged, and the <arglist> is the rest of the command line that is used when <filespec> is invoked and loaded into memory.

CHAPTER 2

COMMANDS

2.1 COMMAND INFORMATION

Each DEBUG command consists of a single letter followed by one or more parameters. Additionally, the control characters and the special editing functions described in the MS-DOS User's Guide, apply inside DEBUG.

If a syntax error occurs in a DEBUG command, DEBUG reprints the command line and indicates the error with an up-arrow (^) and the word "error."

For example:

```
dcs:100 cs:110
    ^ error
```

Any combination of uppercase and lowercase letters may be used in commands and parameters.

The DEBUG commands are summarized in Table 2.1 and are described in detail, with examples, following the description of command parameters.

Table 2.1 DEBUG Commands

DEBUG Command	Function
A[<address>]	Assemble
C<range> <address>	Compare
D[<range>]	Dump
E<address> [<list>]	Enter
F<range> <list>	Fill
G[=<address> [<address>...]]	Go
H<value> <value>	Hex
I<value>	Input
L[<address> [<drive><record><record>]]	Load
M<range> <address>	Move
N<filename>[<filename>]	Name
O<value> <byte>	Output
Q	Quit
R[<register-name>]	Register
S<range> <list>	Search
T[=<address>][<value>]	Trace
U[<range>]	Unassemble
W[<address> [<drive><record><record>]]	Write

2.2 PARAMETERS

All DEBUG commands accept parameters, except the Quit command. Parameters may be separated by delimiters (spaces or commas), but a delimiter is required only between two consecutive hexadecimal values. Thus, the following commands are equivalent:

```
dcs:100 110
d cs:100 110
d,cs:100,110
```

PARAMETER	DEFINITION
<drive>	A one-digit hexadecimal value to indicate which drive a file will be loaded from or written to. The valid values are 0-3. These values designate the drives as follows: 0=A:, 1=B:, 2=C:, 3=D:.
<byte>	A two-digit hexadecimal value to be placed in or read from an address or register.
<record>	A 1- to 3-digit hexadecimal value used to indicate the logical record number on the disk and the number of disk sectors to be written or loaded. Logical records correspond to sectors. However, their numbering differs since they represent the entire disk space.
<value>	A hexadecimal value up to four digits used to specify a port number or the number of times a command should repeat its functions.
<address>	A two-part designation consisting of either an alphabetic segment register designation or a four-digit segment address plus an offset value. The segment designation or segment address may be omitted, in which case the default segment is used. DS is the default segment for all commands except G, L, T, U, and W, for which the default segment is CS. All numeric values are hexadecimal. For example: CS:0100 04BA:0100 The colon is required between a segment designation (whether numeric or alphabetic) and an offset.

<range> Two <address>es: e.g., <address> <address>; or one <address>, an L, and a <value>: e.g., <address> L <value> where <value> is the number of lines the command should operate on, and L80 is assumed. The last form cannot be used if another hex value follows the <range>, since the hex value would be interpreted as the second <address> of the <range>.

Examples:

```
CS:100 110
CS:100 L 10
CS:100
```

The following is illegal:

```
CS:100 CS:110
       ^ error
```

The limit for <range> is 10000 hex. To specify a <value> of 10000 hex within four digits, type 0000 (or 0).

<list> A series of <byte> values or of <string>s. <list> must be the last parameter on the command line.

Example:

```
fcs:100 42 45 52 54 41
```

<string> Any number of characters enclosed in quote marks. Quote marks may be either single (') or double("). If the delimiter quote marks must appear within a <string>, the quote marks must be doubled. For example, the following strings are legal:

```
'This is a "string" is okay.'
'This is a ''string'' is okay.'
```

However, this string is illegal:

```
'This is a 'string' is not.'
```

Similarly, these strings are legal:

```
"This is a 'string' is okay."
"This is a ""string"" is okay."
```

However, this string is illegal:

```
"This is a "string" is not."
```

Note that the double quote marks are not necessary in the following strings:

```
"This is a ''string'' is not necessary."
'This is a ""string"" is not necessary.'
```

The ASCII values of the characters in the string are used as a <list> of byte values.

NAME

Assemble

PURPOSE

Assembles 8086/8087/8088 mnemonics directly into memory.

SYNTAX

A[<address>]

COMMENTS

If a syntax error is found, DEBUG responds with

^Error

and redisplays the current assembly address.

All numeric values are hexadecimal and must be entered as 1-4 characters. Prefix mnemonics must be specified in front of the opcode to which they refer. They may also be entered on a separate line.

The segment override mnemonics are CS:, DS:, ES:, and SS:. The mnemonic for the far return is RETF. String manipulation mnemonics must explicitly state the string size. For example, use MOVSW to move word strings and MOVSB to move byte strings.

The assembler will automatically assemble short, near or far jumps and calls, depending on byte displacement to the destination address. These may be overridden with the NEAR or FAR prefix. For example:

```
0100:0500 JMP   502          ; a 2-byte short jump
0100:0502 JMP   NEAR 505     ; a 3-byte near jump
0100:505  JMP   FAR  50A     ; a 5-byte far jump
```

The NEAR prefix may be abbreviated to NE, but the FAR prefix cannot be abbreviated.

DEBUG cannot tell whether some operands refer to a word memory location or to a byte memory location. In this case, the data type must be explicitly stated with the prefix "WORD PTR" or "BYTE PTR". Acceptable abbreviations are "WO" and "BY". For example:

```
NEG     BYTE PTR [128]
DEC     WO [SI]
```

DEBUG also cannot tell whether an operand refers to a memory location or to an immediate operand. DEBUG uses the common convention that operands enclosed in square brackets refer to memory. For example:

```
MOV     AX,21           ; Load AX with 21H
MOV     AX,[21]         ; Load AX with the
                        ; contents
                        ; of memory location 21H
```

Two popular pseudo-instructions are available with Assemble. The DB opcode will assemble byte values directly into memory. The DW opcode will assemble word values directly into memory. For example:

```
DB      1,2,3,4,"THIS IS AN EXAMPLE"
DB      'THIS IS A QUOTE: "'
DB      "THIS IS A QUOTE: '"

DW      1000,2000,3000,"BACH"
```

Assemble supports all forms of register indirect commands. For example:

```
ADD     BX,34[BP+2].[SI-1]
POP     [BP+DI]
PUSH    [SI]
```

All opcode synonyms are also supported. For example:

```
LOOPZ   100
LOOPE   100

JA      200
JNBE    200
```

For 8087 opcodes, the WAIT or FWAIT must be explicitly specified. For example:

```
FWAIT FADD ST,ST(3)     ; This line will assemble
                        ; an FWAIT prefix
LD TBYTE PTR [BX]       ; This line will not
```

NAME

Compare

PURPOSE

Compares the portion of memory specified by <range> to a portion of the same size beginning at <address>.

SYNTAX

C<range> <address>

COMMENTS

If the two areas of memory are identical, there is no display and DEBUG returns with the MS-DOS prompt. If there are differences, they are displayed in this format:

<address1> <byte1> <byte2> <address2>

EXAMPLE

The following commands have the same effect:

C100,1FF 300

or

C100L100 300

Each command compares the block of memory from 100 to 1FFH with the block of memory from 300 to 3FFH.

NAME

Dump

PURPOSE

Displays the contents of the specified region of memory.

SYNTAX

D[<range>]

COMMENTS

If a range of addresses is specified, the contents of the range are displayed. If the D command is typed without parameters, 128 bytes are displayed at the first address (DS:100) after the address displayed by the previous Dump command.

The dump is displayed in two portions: a hexadecimal dump (each byte is shown in hexadecimal value) and an ASCII dump (the bytes are shown in ASCII characters). Nonprinting characters are denoted by a period (.) in the ASCII portion of the display. Each display line shows 16 bytes with a hyphen between the eighth and ninth bytes. At times, displays are split in this manual to fit them on the page. Each displayed line begins on a 16-byte boundary.

If you type the command:

```
dcs:100 110
```

DEBUG displays the dump in the following format:

```
04BA:0100 42 45 52 54 41 ... 4E 44 TOM SAWYER
```

If you type the following command:

```
D
```

the display is formatted as described above. Each line of the display begins with an address, incremented by 16 from the address on the previous line. Each subsequent D (typed without parameters) displays the bytes immediately following those last displayed.

If you type the command:

```
DCS:100 L 20
```

the display is formatted as described above, but 20H bytes are displayed.

If then you type the command:

```
DCS:100 115
```

the display is formatted as described above, but all the bytes in the range of lines from 100H to 115H in the CS segment are displayed.

NAME

Enter

PURPOSE

Enters byte values into memory at the specified <address>.

SYNTAX

E<address>[<list>]

COMMENTS

If the optional <list> of values is typed, the replacement of byte values occurs automatically. (If an error occurs, no byte values are changed.)

If the <address> is typed without the optional <list>, DEBUG displays the address and its contents, then repeats the address on the next line and waits for your input. At this point, the Enter command waits for you to perform one of the following actions:

1. Replace a byte value with a value you type. Simply type the value after the current value. If the value typed in is not a legal hexadecimal value or if more than two digits are typed, the illegal or extra character is not echoed.

2. Press the <SPACE> bar to advance to the next byte. To change the value, simply type the new value as described in (1.) above. If you space beyond an 8-byte boundary, DEBUG starts a new display line with the address displayed at the beginning.

3. Type a hyphen (-) to return to the preceding byte. If you decide to change a byte behind the current position, typing the hyphen returns the current position to the previous byte. When the hyphen is typed, a new line is started with the address and its byte value displayed.

4. Press the <RETURN> key to terminate the Enter command. The <RETURN> key may be pressed at any byte position.

EXAMPLE

Assume that the following command is typed:

```
ECS:100
```

DEBUG displays:

```
04BA:0100  EB._
```

To change this value to 41, type 41 as shown:

```
04BA:0100  EB.41_
```

To step through the subsequent bytes, press the <SPACE> bar to see:

```
04BA:0100  EB.41    10.    00.    BC._
```

To change BC to 42:

```
04BA:0100  EB.41    10.    00.    BC.42_
```

Now, realizing that 10 should be 6F, type the hyphen as many times as needed to return to byte 0101 (value 10), then replace 10 with 6F:

```
04BA:0100  EB.41    10.    00.    BC.42-
04BA:0102  00.-_
04BA:0101  10.6F_
```

Pressing the <RETURN> key ends the Enter command and returns to the DEBUG command level.

NAME

Fill

PURPOSE

Fills the addresses in the <range> with the values in the <list>.

SYNTAX

F<range> <list>

COMMENTS

If the <range> contains more bytes than the number of values in the <list>, the <list> will be used repeatedly until all bytes in the <range> are filled. If the <list> contains more values than the number of bytes in the <range>, the extra values in the <list> will be ignored. If any of the memory in the <range> is not valid (bad or nonexistent), the error will occur in all succeeding locations.

EXAMPLE

Assume that the following command is typed:

```
F04BA:100 L 100 42 45 52 54 41
```

DEBUG fills memory locations 04BA:100 through 04BA:1FF with the bytes specified. The five values are repeated until all 100H bytes are filled.

NAME

Go

PURPOSE

Executes the program currently in memory.

SYNTAX

G[=<address>[<address>...]]

COMMENTS

If only the Go command is typed, the program executes as if the program had run outside DEBUG.

If =<address> is set, execution begins at the address specified. The equal sign (=) is required, so that DEBUG can distinguish the start =<address> from the breakpoint <address>es.

With the other optional addresses set, execution stops at the first <address> encountered, regardless of that address' position in the list of addresses to halt execution or program branching. When program execution reaches a breakpoint, the registers, flags, and decoded instruction are displayed for the last instruction executed. (The result is the same as if you had typed the Register command for the breakpoint address.)

Up to ten breakpoints may be set. Breakpoints may be set only at addresses containing the first byte of an 8086 opcode. If more than ten breakpoints are set, DEBUG returns the BP Error message.

The user stack pointer must be valid and have 6 bytes available for this command. The G command uses an IRET instruction to cause a jump to the program under test. The user stack pointer is set, and the user flags, Code Segment register, and Instruction Pointer are pushed on the user stack. (Thus, if the user stack is not valid or is too small, the operating system may crash.) An interrupt code (0CCH) is placed at the specified breakpoint address(es).

When an instruction with the breakpoint code is encountered, all breakpoint addresses are restored to their original instructions. If

execution is not halted at one of the breakpoints, the interrupt codes are not replaced with the original instructions.

EXAMPLE

Assume that the following command is typed:

```
GCS:7550
```

The program currently in memory executes up to the address 7550 in the CS segment. DEBUG then displays registers and flags, after which the Go command is terminated.

After a breakpoint has been encountered, if you type the Go command again, then the program executes just as if you had typed the filename at the MS-DOS command level. The only difference is that program execution begins at the instruction after the breakpoint rather than at the usual start address.

NAME

Hex

PURPOSE

Performs hexadecimal arithmetic on the two parameters specified.

SYNTAX

H<value> <value>

COMMENTS

First, DEBUG adds the two parameters, then subtracts the second parameter from the first. The results of the arithmetic are displayed on one line; first the sum, then the difference.

EXAMPLE

Assume that the following command is typed:

```
H19F 10A
```

DEBUG performs the calculations and then displays the result:

```
02A9   0095
```

NAME

Input

PURPOSE

Inputs and displays one byte from the port specified by <value>.

SYNTAX

I<value>

COMMENTS

A 16-bit port address is allowed.

EXAMPLE

Assume that you type the following command:

```
I2F8
```

Assume also that the byte at the port is 42H. DEBUG inputs the byte and displays the value:

```
42
```

NAME

Load

PURPOSE

Loads a file into memory.

SYNTAX

L[<address> [<drive> <record> <record>]]

COMMENTS

Set BX:CX to the number of bytes read. The file must have been named either when DEBUG was started or with the N command. Both the DEBUG invocation and the N command format a filename properly in the normal format of a file control block at CS:5C.

If the L command is typed without any parameters, DEBUG loads the file into memory beginning at address CS:100 and sets BX:CX to the number of bytes loaded. If the L command is typed with an address parameter, loading begins at the memory <address> specified. If L is typed with all parameters, absolute disk sectors are loaded, not a file. The <record>s are taken from the <drive> specified (the drive designation is numeric here--0=A:, 1=B:, 2=C:, etc.); DEBUG begins loading with the first <record> specified, and continues until the number of sectors specified in the second <record> have been loaded.

EXAMPLE

Assume that the following commands are typed:

```
A>DEBUG
-NFILE.COM
```

Now, to load FILE.COM, type:

```
L
```

DEBUG loads the file and then displays the DEBUG prompt. Assume that you want to load only portions of a file or certain records from a disk. To do this, type:

```
L04BA:100 2 0F 6D
```

DEBUG then loads 109 (6D hex) records beginning with logical record number 15 into memory

beginning at address 04BA:0100. When the records have been loaded, DEBUG simply returns the - prompt.

If the file has a .EXE extension, it is relocated to the load address specified in the header of the .EXE file: the <address> parameter is always ignored for .EXE files. The header itself is stripped off the .EXE file before it is loaded into memory. Thus the size of an .EXE file on disk will differ from its size in memory.

If the file named by the Name command or specified when DEBUG is started is a .HEX file, then typing the L command with no parameters causes DEBUG to load the file beginning at the address specified in the .HEX file. If the L command includes the option <address>, DEBUG adds the <address> specified in the L command to the address found in the .HEX file to determine the start address for loading the file.

NAME

Move

PURPOSE

Moves the block of memory specified by <range> to the location beginning at the <address> specified.

SYNTAX

M<range> <address>

COMMENTS

Overlapping moves (i.e., moves where part of the block overlaps some of the current addresses) are always performed without loss of data. Addresses that could be overwritten are moved first. The sequence for moves from higher addresses to lower addresses is to move the data beginning at the block's lowest address and then to work towards the highest. The sequence for moves from lower addresses to higher addresses is to move the data beginning at the block's highest address and to work towards the lowest.

Note that if the addresses in the block being moved will not have new data written to them, the data there before the move will remain. The M command copies the data from one area into another, in the sequence described, and writes over the new addresses. This is why the sequence of the move is important.

EXAMPLE

Assume that you type:

```
MCS:100 110 CS:500
```

DEBUG first moves address CS:110 to address CS:510, then CS:10F to CS:50F, and so on until CS:100 is moved to CS:500. You should type the D command, using the <address> typed for the M command, to review the results of the move.

NAME

Name

PURPOSE

Sets filenames.

SYNTAX

N<filename>[<filename>...]

COMMENTS

The Name command performs two functions. First, Name is used to assign a filename for a later Load or Write command. Thus, if you start DEBUG without naming any file to be debugged, then the N<filename> command must be typed before a file can be loaded. Second, Name is used to assign filename parameters to the file being debugged. In this case, Name accepts a list of parameters that are used by the file being debugged.

These two functions overlap. Consider the following set of DEBUG commands:

```
-NFILE1.EXE
-L
-G
```

Because of the effects of the Name command, Name will perform the following steps:

1. (N)ame assigns the filename FILE1.EXE to the filename to be used in any later Load or Write commands.

2. (N)ame also assigns the filename FILE1.EXE to the first filename parameter used by any program that is later debugged.

3. (L)oad loads FILE1.EXE into memory.

4. (G)o causes FILE1.EXE to be executed with FILE1.EXE as the single filename parameter (that is, FILE1.EXE is executed as if FILE1.EXE had been typed at the command level).

A more useful chain of commands might look like this:

```
-NFILE1.EXE
-L
-NFILE2.DAT FILE3.DAT
-G
```

Here, Name sets FILE1.EXE as the filename for the subsequent Load command. The Load command loads FILE1.EXE into memory, and then the Name command is used again, this time to specify the parameters to be used by FILE1.EXE. Finally, when the Go command is executed, FILE1.EXE is executed as if FILE1 FILE2.DAT FILE3.DAT had been typed at the MS-DOS command level. Note that if a Write command were executed at this point, then FILE1.EXE--the file being debugged--would be saved with the name FILE2.DAT! To avoid such undesired results, you should always execute a Name command before either a Load or a Write.

There are four regions of memory that can be affected by the Name command:

```
CS:5C  FCB for file 1
CS:6C  FCB for file 2
CS:80  Count of characters
CS:81  All characters typed
```

A File Control Block (FCB) for the first filename parameter given to the Name command is set up at CS:5C. If a second filename parameter is typed, then an FCB is set up for it beginning at CS:6C. The number of characters typed in the Name command (exclusive of the first character, "N") is given at location CS:80. The actual stream of characters given by the Name command (again, exclusive of the letter "N") begins at CS:81. Note that this stream of characters may contain switches and delimiters that would be legal in any command typed at the MS-DOS command level.

EXAMPLE

A typical use of the Name command is:

```
DEBUG PROG.COM
-NPARAM1 PARAM2/C
-G
-
```

In this case, the Go command executes the file in memory as if the following command line had been typed:

```
PROG PARAM1 PARAM2/C
```

Testing and debugging therefore reflect a normal runtime environment for PROG.COM.

NAME

Output

PURPOSE

Sends the <byte> specified to the output port specified by <value>.

SYNTAX

O<value> <byte>

COMMENTS

A 16-bit port address is allowed.

EXAMPLE

Type:

```
O2F8 4F
```

DEBUG outputs the byte value 4F to output port 2F8.

NAME

Quit

PURPOSE

Terminates the DEBUG utility.

SYNTAX

Q

COMMENTS

The Q command takes no parameters and exits DEBUG without saving the file currently being operated on. You are returned to the MS-DOS command level.

EXAMPLE

To end the debugging session, type:

```
Q<RETURN>
```

DEBUG has been terminated, and control returns to the MS-DOS command level.

NAME

Register

PURPOSE

Displays the contents of one or more CPU registers.

SYNTAX

R[<register-name>]

COMMENTS

If no <register-name> is typed, the R command dumps the register save area and displays the contents of all registers and flags.

If a register name is typed, the 16-bit value of that register is displayed in hexadecimal, and then a colon appears as a prompt. You then either type a <value> to change the register, or simply press the <RETURN> key if no change is wanted.

The only valid <register-name>s are:

AX	BP	SS	
BX	SI	CS	
CX	DI	IP	(IP and PC both refer
DX	DS	PC	to the Instruction
SP	ES	F	Pointer.)

Any other entry for <register-name> results in a BR Error message.

If F is entered as the <register-name>, DEBUG displays each flag with a two-character alphabetic code. To alter any flag, type the opposite two-letter code. The flags are either set or cleared.

The flags are listed below with their codes for SET and CLEAR:

FLAG NAME	SET	CLEAR
Overflow	OV	NV
Direction	DN Decrement	UP Increment
Interrupt	EI Enabled	DI Disabled
Sign	NG Negative	PL Plus
Zero	ZR	NZ
Auxiliary Carry	AC	NA
Parity	PE Even	PO Odd
Carry	CY	NC

Whenever you type the command RF, the flags are displayed in the order shown above in a row at the beginning of a line. At the end of the list of flags, DEBUG displays a hyphen (-). You may enter new flag values as alphabetic pairs. The new flag values can be entered in any order. You do not have to leave spaces between the flag entries. To exit the R command, press the <RETURN> key. Flags for which new values were not entered remain unchanged.

If more than one value is entered for a flag, DEBUG returns a DF Error message. If you enter a flag code other than those shown above, DEBUG returns a BF Error message. In both cases, the flags up to the error in the list are changed; flags at and after the error are not.

At startup, the segment registers are set to the bottom of free memory, the Instruction Pointer is set to 0100H, all flags are cleared, and the remaining registers are set to zero.

EXAMPLE

Type:

```
R
```

DEBUG displays all registers, flags, and the decoded instruction for the current location. If the location is CS:11A, then the display will look similar to this:

```
AX=0E00 BX=00FF CX=0007 DX=01FF SP=039D BP=0000
SI=005C DI=0000 DS=04BA ES=04BA SS=04BA CS=04BA
IP=011A  NV UP DI NG NZ AC PE NC
04BA:011A  CD21          INT     21
```

If you type:

```
RF
```

DEBUG will display the flags:

```
NV UP DI NG NZ AC PE NC - _
```

Now, type any valid flag designation, in any order, with or without spaces.

For example:

```
NV UP DI NG NZ AC PE NC - PLEICY<RETURN>
```

DEBUG responds only with the DEBUG prompt. To see the changes, type either the R or RF command:

```
RF
NV UP EI PL NZ AC PE CY - _
```

Press <RETURN> to leave the flags this way, or to specify different flag values.

NAME

Search

PURPOSE

Searches the <range> specified for the <list> of bytes specified.

SYNTAX

S<range> <list>

COMMENTS

The <list> may contain one or more bytes, each separated by a space or comma. If the <list> contains more than one byte, only the first address of the byte string is returned. If the <list> contains only one byte, all addresses of the byte in the <range> are displayed.

EXAMPLE

If you type:

```
SCS:100 110 41
```

DEBUG will display a response similar to this:

```
04BA:0104
04BA:010D
-type:
```

NAME

Trace

PURPOSE

Executes one instruction and displays the contents of all registers and flags, and the decoded instruction.

SYNTAX

T[=<address>][<value>]

COMMENTS

If the optional =<address> is typed, tracing occurs at the =<address> specified. The optional <value> causes DEBUG to execute and trace the number of steps specified by <value>.

The T command uses the hardware trace mode of the 8086 or 8088 microprocessor. Consequently, you may also trace instructions stored in ROM (Read Only Memory).

EXAMPLE

Type:

```
T
```

DEBUG returns a display of the registers, flags, and decoded instruction for that one instruction. Assume that the current position is 04BA:011A; DEBUG might return the display:

```
AX=0E00 BX=00FF CX=0007 DX=01FF SP=039D BP=0000
SI=005C DI=0000 DS=04BA ES=04BA SS=04BA CS=04BA
IP=011A  NV UP DI NG NZ AC PE NC
04BA:011A  CD21          INT     21
```

If you type

```
T=011A 10
```

DEBUG executes sixteen (10 hex) instructions beginning at 011A in the current segment, and then displays all registers and flags for each instruction as it is executed. The display scrolls away until the last instruction is executed. Then the display stops, and you can see the register and flag values for the last few instructions performed. Remember that <CONTROL-S> suspends the display at any point, so that you can study the registers and flags for any instruction.

NAME

Unassemble

PURPOSE

Disassembles bytes and displays the source statements that correspond to them, with addresses and byte values.

SYNTAX

U[<range>]

COMMENTS

The display of disassembled code looks like a listing for an assembled file. If you type the U command without parameters, 20 hexadecimal bytes are disassembled at the first address after that displayed by the previous Unassemble command. If you type the U command with the <range> parameter, then DEBUG disassembles all bytes in the range. If the <range> is given as an <address> only, then 20H bytes are disassembled instead of 80H.

EXAMPLE

Type:

```
U04BA:100 L10
```

DEBUG disassembles 16 bytes beginning at address 04BA:0100:

```
04BA:0100  206472    AND  [SI+72],AH
04BA:0103  69        DB   69
04BA:0104  7665      JBE  016B
04BA:0106  207370    AND  [BP+DI+70],DH
04BA:0109  65        DB   65
04BA:010A  63        DB   63
04BA:010B  69        DB   69
04BA:010C  66        DB   66
04BA:010D  69        DB   69
04BA:010E  63        DB   63
04BA:010F  61        DB   61
```

If you type

```
U04ba:0100 0108
```

The display will show:

```
04BA:0100  206472      AND   [SI+72],AH
04BA:0103  69          DB    69
04BA:0104  7665        JBE   016B
04BA:0106  207370      AND   [BP+DI+70],DH
```

If the bytes in some addresses are altered, the disassembler alters the instruction statements. The U command can be typed for the changed locations, the new instructions viewed, and the disassembled code used to edit the source file.

NAME

Write

PURPOSE

Writes the file being debugged to a disk file.

SYNTAX

W[<address>[<drive> <record> <record>]]

COMMENTS

If you type W with no parameters, BX:CX must already be set to the number of bytes to be written; the file is written beginning from CS:100. If the W command is typed with just an address, then the file is written beginning at that address. If a G or T command has been used, BX:CX must be reset before using the Write command without parameters. Note that if a file is loaded and modified, the name, length, and starting address are all set correctly to save the modified file (as long as the length has not changed).

The file must have been named either with the DEBUG invocation command or with the N command (refer to the Name command earlier in this manual). Both the DEBUG invocation and the N command format a filename properly in the normal format of a file control block at CS:5C.

If the W command is typed with parameters, the write begins from the memory address specified; the file is written to the <drive> specified (the drive designation is numeric here--0=A:, 1=B:, 2=C:, etc.); DEBUG writes the file beginning at the logical record number specified by the first <record>; DEBUG continues to write the file until the number of sectors specified in the second <record> have been written.

WARNING

Writing to absolute sectors is EXTREMELY dangerous because the process bypasses the file handler.

EXAMPLE

Type:

```
W
```

DEBUG will write the file to disk and then display the DEBUG prompt. Two examples are shown below.

```
W
-_
```

```
WCS:100 1 37 2B
```

DEBUG writes out the contents of memory, beginning with the address CS:100 to the disk in drive B:. The data written out starts in disk logical record number 37H and consists of 2BH records. When the write is complete, DEBUG displays the prompt:

```
WCS:100 1 37 2B
-_
```

2.3 ERROR MESSAGES

During the DEBUG session, you may receive any of the following error messages. Each error terminates the DEBUG command under which it occurred, but does not terminate DEBUG itself.

ERROR CODE	DEFINITION
BF	Bad flag You attempted to alter a flag, but the characters typed were not one of the acceptable pairs of flag values. See the Register command for the list of acceptable flag entries.
BP	Too many breakpoints You specified more than ten breakpoints as parameters to the G command. Retype the Go command with ten or fewer breakpoints.
BR	Bad register You typed the R command with an invalid register name. See the Register command for the list of valid register names.
DF	Double flag You typed two values for one flag. You may specify a flag value only once per RF command.

INDEX

DEBUG Commands
(A)ssemble 2-6
(C)ompare 2-8
(D)ump 2-9
(E)nter 2-11
(Fill) 2-13
(G)o 2-14
(H)ex 2-16
(I)nput 2-17
(L)oad 2-18
(M)ove 2-19
(N)ame 2-21
(O)utput 2-24
(Q)uit 2-25
(R)egister 2-26
(S)earch 2-29
(T)race 2-30
(U)nassemble 2-32
(W)rite 2-34
DEBUG Errors
BF - Bad flag 2-36
BP - Too many breakpoints . 2-36
BR - Bad register 2-36
DF - Double flag 2-36

EXE files 2-19

Flags 2-27